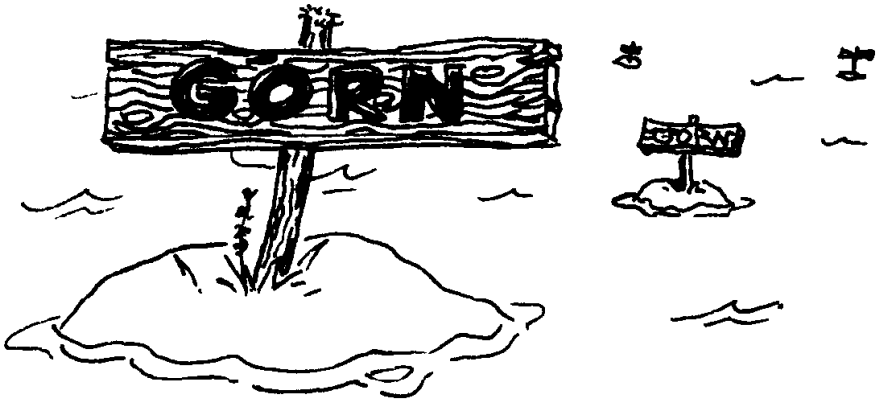
Projekt Nebenläufige Programmierung in Erlang
Sommersemester 2001

Gorn: Gorn On Random Nodes

Raimund Jacob <raimi@cs.tu-berlin.de>
André Seidelt <iluvatar@cs.tu-berlin.de>
Martin Grabmüller <mgrabmue@cs.tu-berlin.de>

2. November 2001



a good, distributed, wooden application

Technische Universität Berlin
Fakultät IV Elektrotechnik und Informatik
Petra Hofstedt

Inhaltsverzeichnis

1 Einleitung **3**

2 Systemarchitektur **3**
2.1 Benutzermodell . . . 3
2.2 Architektur . . . 3
2.3 GEF: Das GORN Executable Format . . . 5
2.4 Die Servermodule . . . 6
2.4.1 Der GORN-Prozess . . . 6
2.4.2 Der Usermanager . . . 7
2.4.3 Der Applicationmanager . . . 9
2.4.4 Der DB-Server . . . 10

3 Applikationen **11**
3.1 Die Gornbar . . . 11
3.2 Das Mailsystem . . . 11
3.3 Das Chatsystem . . . 13
3.4 Das Talksystem . . . 14
3.5 Die Verwaltungsapplikation . . . 16
3.6 Die Spiele . . . 16
3.6.1 Der Gameserver . . . 17
3.6.2 Das `game`-Modul . . . 17
3.6.3 GORNtris . . . 19
3.6.4 Pong . . . 21
3.6.5 Tron . . . 25
3.7 Die About-Box . . . 26
3.8 Die Bibliothek `libgorn` . . . 26

4 Fazit **28**
4.1 Die Programmiersprache ERLANG . . . 28
4.2 Bewertung der Arbeitsergebnisse . . . 30

A Benutzerhandbuch **32**
A.1 Der GORN Server . . . 32
A.2 Die Gornbar . . . 33
A.3 Email . . . 33
A.4 Chat . . . 34
A.5 Talk . . . 34
A.6 Admin . . . 34
A.7 Spiele: GORNtris, Tron, Pong . . . 35
A.7.1 GORNtris spielen . . . 35
A.7.2 GORNtris konfigurieren . . . 36
A.7.3 Pong spielen . . . 36
A.7.4 Pong konfigurieren . . . 36
A.7.5 Tron spielen . . . 37
A.7.6 Tron konfigurieren . . . 37
A.8 About-Box . . . 37

1 Einleitung

Diese Ausarbeitung begleitet die Applikation Gorn, ein verteiltes Kommunikationssystem für den Einsatz in kleinen bis mittleren Arbeitsgruppen. Die Arbeit soll als Dokumentation des Programms und als Benutzerhandbuch dienen. Ziel des Projektes war, unter Verwendung der Programmiersprache Erlang, eine verteilte, nebenläufige und kommunikationsbasierte Anwendung zu entwickeln.

In Abschnitt 2 werden die grundlegende Architektur unseres Systems sowie die Servermodule vorgestellt. Weiterhin sollen die Systemkomponenten näher erläutert und die Kommunikation zwischen einzelnen Prozessen beschrieben werden. Im darauf folgenden Abschnitt 3 beschreiben wir die Applikationen des Gorn-Systems.

Abschließend werden die Autoren in Abschnitt 4 ihre Erfahrungen mit der funktionalen Programmiersprache Erlang darlegen sowie eine Bewertung des Projektergebnisses und des Erlang-Systems vornehmen. Zusätzlich haben wir einzelne Komponenten bereits im Text bewertet. Der Anhang umfasst Beschreibungen zur Installation und Benutzung des Systems (Anhang A).

2 Systemarchitektur

Dieser Abschnitt umfasst eine Beschreibung der Systemarchitektur und eine präzise Dokumentation der einzelnen Servermodule.

2.1 Benutzermodell

Eine Instanz des Gorn Systems stellt eine „Arbeitsgruppe“ dar. Dabei besteht eine Instanz aus einem Serverknoten, auf dem die Serverprozesse laufen, sowie beliebig vielen Clientrechnern, die die Clientprozesse ausführen.

In der Arbeitsgruppe werden eine Menge von Benutzern verwaltet, die eindeutig über ihren Benutzernamen identifiziert sind. Jedem verbundenen Client sind alle Benutzer bekannt. Benutzer können privilegiert sein und somit mehr Rechte bei der Administration des Gesamtsystems haben. Beispielsweise können nur sie Benutzerkonten anlegen und löschen.

Die Informationen über einen Benutzer bestehen aus einem privaten und einem öffentlichen Teil. Der öffentliche Teil der Benutzerdaten kann von jedem Gruppenmitglied gelesen werden. Der private Teil kann nur vom selben Benutzer und von privilegierten Benutzern gelesen und verändert werden. Zum privaten Teil gehört z.B. das Passwort. Der öffentliche Teil umfasst den Benutzernamen, den Realnamen und den aktuellen Zustand (`online`, `offline`, `away`). `Offline` sind alle Benutzer, die zwar im System existieren, aber nicht eingeloggt sind. `Online` und `away` sind eingeloggte Benutzer, wobei `away` ein spezieller Zustand ist, mit dem signalisiert wird, dass der Benutzer sich vorübergehend vom Arbeitsplatz entfernt hat.

Bewertung

Sowohl öffentlicher als auch privater Teil der Benutzerinformationen sind erweiterbar. Der öffentliche Teil könnte beispielsweise noch organisationsinterne Adressen und Telefonnummern enthalten. Der private Teil könnte noch um vertrauliche Personaldaten wie Kontonummern erweitert werden.

Das zweistufige Benutzerkonzept ist sehr einfach und nur für kleine Arbeitsgruppen geeignet. In größeren Organisationen wäre eine ausgefeilte Rechteverwaltung nötig. In solchen Organisationen wäre es sinnvoll, die Benutzerverwaltung an existierende Systeme anzubinden.

2.2 Architektur

Gorn basiert auf einer Client-Server-Architektur, deren Komponenten in Abbildung 1 dargestellt sind. Es gibt einige Serverprozesse, die auf einem zentralen Rechner laufen, der allen Clients bekannt sein muss. Diese Serverprozesse stellen verschiedene Dienste zur Verfügung. Zum einen gibt es Server, die der Benutzer-, Applikations- und Datenbankverwaltung dienen, zum anderen

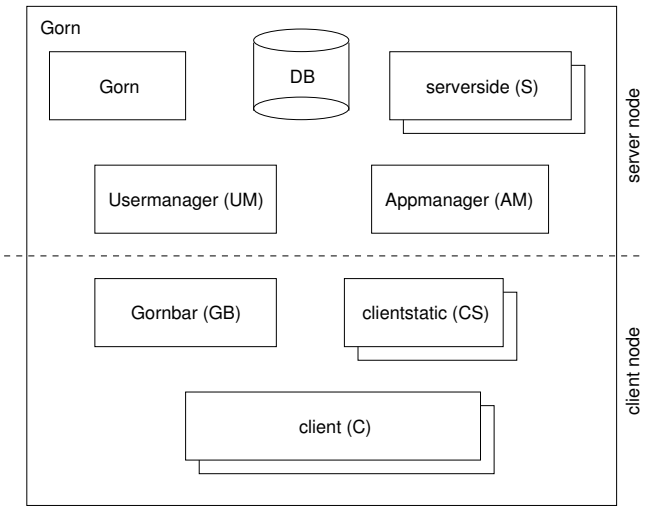


Abbildung 1: Systemarchitektur

stellen applikationsspezifische Server den Clientapplikationen zentrale Dienste zur Verfügung, zum Beispiel die Verwaltung von Email-Nachrichten.

Auf der Clientseite laufen verschiedene Arten von Prozessen. Der Wichtigste davon ist die Gornbar, die als einziges Modul auf den Clientrechnern installiert werden muss. Diese hat allerdings nur eine sehr eingeschränkte Funktionalität: Im Wesentlichen kann sie das Einloggen auf dem Server, eine Liste aller Benutzer und ein Nachrichtenfenster verwalten. Die Applikationen selbst, die die eigentliche Funktionalität zur Verfügung stellen, werden nicht auf der Clientseite installiert, sie existieren nur auf dem Server und werden bei Bedarf in den ERLANG-Prozess auf dem Clientrechner geladen. Dieses dynamische Laden macht die Installation und Konfiguration des Systems sehr einfach, da auf den Clients lediglich die Gornbar installiert werden muss. Die Applikationen selbst müssen nur auf dem Server eingespielt werden und stehen den Benutzern beim nächsten Einloggen zur Verfügung. Wie das dynamische Laden von Applikationen implementiert wurde, ist in Abschnitt 2.3 beschrieben.

Die Applikationen können aus bis zu drei unabhängigen Teilen bestehen (siehe Abbildung 2). Die eigentliche Clientapplikation implementiert die grafische Benutzeroberfläche und die komplette Logik zur Benutzerinteraktion. Sie wird explizit vom Benutzer gestartet. Dieser Teil der Applikation wird im Folgenden als *Client* bezeichnet. Applikationen können, falls nötig, noch einen serverseitigen Teil besitzen. Dieser wird als *Serverside* bezeichnet. Und schließlich darf noch eine so genannte *Clientstatic* Komponente der Applikation existieren, die beim Einloggen auf dem Clientrechner gestartet und erst beim Ausloggen wieder beendet wird. Beispiele für diese drei Komponenten werden in der Dokumentation der einzelnen Applikationen gegeben.

Die Dokumentation der einzelnen Systemkomponenten beinhaltet jeweils eine Referenz der wichtigsten GORN-spezifischen Nachrichten, die von der Komponente verschickt bzw. empfangen werden. Dabei werden folgende Abkürzungen zur Bezeichnung der Systembestandteile benutzt: **C** ist die allgemeine Abkürzung für Clients, **S** bezeichnet einen Server. Spezielle Server und Clients werden ihren Namen entsprechend abgekürzt. **UM** ist der Usermanager, **AM** der Applicationmanager und **G** der GORN-Prozess. Die Gornbar wird mit **GB** abgekürzt.

Bewertung

Das Applikationsmodell ist sehr leicht erweiterbar. Neue Applikationen müssen nur im richtigen Verzeichnis verfügbar gemacht werden. Negativ ist zu bewerten, dass der GORN Server neu gestartet werden muss, um neue Applikationen zu finden. Allerdings ist das Erneuern von existierenden Applikationen sehr einfach. Ein neu übersetztes Modul wird automatisch nachgeladen, so dass

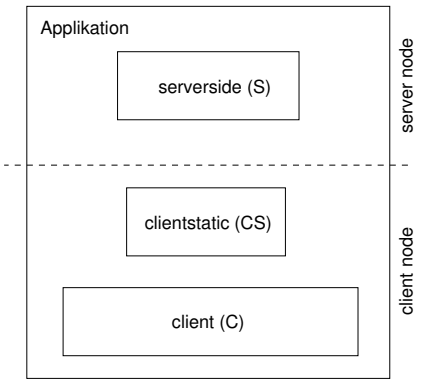


Abbildung 2: Bestandteile einer Applikation

immer die aktuellste Version gestartet wird.

2.3 GEF: Das Gorn Executable Format

Das Gorn Executable Format beschreibt ein bestimmtes Interface, das Serversides, Clients und Clientstatics einer Applikation implementieren müssen, um vom Gorn-System akzeptiert zu werden. Das Interface umfasst die Signaturen der zu exportierenden Funktionen sowie das Protokoll zum Laden und Starten der Komponenten. Die Signaturen werden vom GEF-Loader geprüft, die korrekte Implementierung des Protokolls allerdings nicht. Da die Module nur auf dem Server installiert werden, stellt dies bei sachgemäßer Installation kein Sicherheitsrisiko dar. Die Modulnamen werden dem GEF als Atome übergeben und müssen eindeutig sein.

Eine Applikation muss eine Funktion `gornapp` mit der Arität eins exportieren und folgende Atome erkennen:

`name:` Liefert eine Zeichenkette mit dem Namen der Applikation. Dieser Name wird in den Knöpfen der Gornbar angezeigt.

`version:` Liefert ein Drei-Tupel mit der Version der Applikation. Diese Information könnte zum Abgleich der Versionen von Server- und Clientmodulen benutzt werden, wird im Moment aber ignoriert.

`linklist:` Liefert eine Liste von Modulen, die von dieser Applikation benötigt werden. Diese werden beim Starten der Applikation mit auf die Clientnode übertragen.

`description:` Liefert eine textuelle Beschreibung der Applikation. Die Beschreibung muss als Liste von Zeichenketten zurückgegeben werden, wobei jedes Listenelement auf einer Zeile angezeigt wird. Dieser Text könnte von einer Hilfefunktion angezeigt werden, wird aber im Moment ignoriert.

Die Applikationen werden vom Applicationmanager auf der Clientnode gestartet. Die folgenden Parameter werden dabei den Clientstatics bzw. Applikationen als Liste übergeben:

`Gornbar:` Prozess-Identifier (PID) der Gornbar, von der die Applikation gestartet wurde.

`Cookie:` Der Cookie des Benutzers, der die Applikation gestartet hat.

`Username:` Der Benutzername des Benutzers, der die Applikation gestartet hat.

`Serverlist:` Eine Liste von Zwei-Tupeln, wobei jedes Tupel aus dem Namen eines Servers und dessen PID besteht. Neben den Applikationsservern beinhaltet die Liste auch die PIDs des Application- und des Usermanagers unter den Namen `appman` bzw. `userman`.

Die Interfaces für Clientstatic- und Servermodule sind wesentlich einfacher gehalten: Statics exportieren die Funktion `gornstatic`, die das Atom `linklist` akzeptiert. Zum Starten des Clientstatics wird die selbe Funktion mit einer Parameterliste aufgerufen. Server exportieren nur die Funktion `gornserv`, die ohne Parameter auskommt.

Die Clientmodule können Kontakt zu ihren Servern aufnehmen und deren Dienste nutzen. Dazu entnehmen sie die PIDs aus der Liste der Server, die sie beim Start übergeben bekommen haben.

Bewertung

Das GEF ermittelt Informationen von den Modulen, die in GORN momentan nicht benutzt werden. Zukünftige Erweiterungen können darauf zurückgreifen.

2.4 Die Servermodule

Auf dem Server laufen nach dem Starten des Systems zunächst vier applikationsunabhängige Server: Der GORN-Prozess, der Usermanager, der Applicationmanager und der DB-Server.

Neben diesen vier Servern, die immer gestartet werden, durchsucht der Applicationmanager beim Starten ein Verzeichnis, welches zu startende Serversidemodule enthalten kann. Diese werden dadurch identifiziert, dass sie gültige Beam-Dateien (ERLANG-Binärdateien) sein müssen, die das GEF-Interface für Serversides exportieren. Alle gefundenen Server werden dann gestartet. Darunter sind beispielsweise der Chatserver und der Mailserver.

2.4.1 Der GORN-Prozess

Der GORN-Prozess ist der erste Prozess des GORN-Systems. Er registriert sich unter dem Namen `gorn` beim ERLANG-Laufzeitsystem. Die Gornbar versucht beim Einloggen des Benutzers Kontakt mit diesem Prozess aufzunehmen und leitet dadurch das Login-Protokoll ein. Da die Anzahl der Prozesse, die den ERLANG-Laufzeitsystemen bekannt sein müssen, gering gehalten werden sollte, ist dieses Protokoll etwas komplizierter (siehe auch Abbildung 3). Es umfasst zwei Stufen: Zunächst sendet die Gornbar die eigene PID, den Usernamen und das eingegebene Passwort an den GORN-Prozess. Wenn die Username/Passwort-Kombination gültig ist, wird der Gornbar der User- und der Application-Server bekannt gemacht und die zweite Stufe beginnt. Sie ist in Abschnitt 2.4.2 beschrieben.

Nach dem Einloggen kommuniziert keiner der clientseitigen Prozesse mehr mit dem GORN-Prozess.

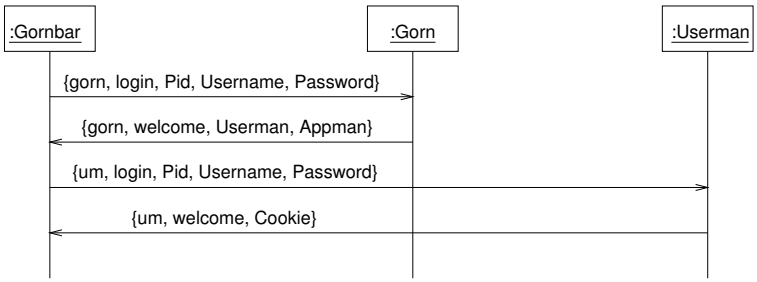


Abbildung 3: Das Login-Protokoll

Nachrichten-Referenz – Gorn-Prozess

GB → **G** {gorn, login, *Pid*, *Username*, *Password*} :
Benutzer *Username* will sich mit der Gornbar *Pid* und *Password* anmelden.

G → GB {gorn, welcome, *PID_Usermanager*, *PID_Appmanager*} :
Benutzer hat sich erfolgreich authentifiziert und bekommt zur weiteren Anmeldung die PIDs der wichtigsten Serverprozesse mitgeteilt.

G → GB {gorn, reject} :
Benutzer darf sich nicht anmelden, weil das Passwort nicht gestimmt hat oder der Benutzer nicht bekannt ist.

G → AM {gorn, serverlist, *Servers*} :
Der GORN-Prozess schickt diese Nachricht an den Applicationmanager, sobald die Servermodule gestartet wurden. Dies ist notwendig, da der Applicationmanager die Liste benötigt, wenn er Clients bzw. Clientstatics startet.

2.4.2 Der Usermanager

Der Usermanager übernimmt die zweite Stufe des Login-Protokolls und ist generell für die Verwaltung der Benutzerdaten zuständig.

Clientprozesse können sich beim Usermanager registrieren, um über Status-Änderungen der GORN-Benutzer informiert zu werden. Serversides werden automatisch registriert. Je nach Applikation wird diese Information dazu verwendet, eine Mitteilung anzuzeigen oder bestimmte Aktionen auszuführen. Ein Beispiel hierfür ist die Gornbar, die diese Änderungen im Logfenster anzeigt.

Eine weitere Funktion, die der Usermanager übernimmt, betrifft den so genannten Watchdogmechanismus. Dabei handelt es sich um ein Nachrichtenprotokoll, das sicherstellt, dass der Server und die Gornbar die Erreichbarkeit des jeweils anderen Prozesses prüfen können.

Die Erreichbarkeit wird überprüft, indem der Usermanager in regelmäßigen Abständen (4 Sekunden) eine Nachricht an alle eingeloggten Gornbars schickt und nach jeweils 2 Sekunden prüft, welche Prozesse geantwortet haben. Sollte ein Prozess nicht in der gegebenen Zeitspanne geantwortet haben, wird er automatisch ausgeloggt. Den Gornbars dient dieser Mechanismus zum Testen, ob der GORN-Server noch erreichbar ist. Sollte dieser für 5 Sekunden keine Watchdog-Nachricht schicken, loggt sich die Gornbar automatisch aus, um in einen definierten Zustand zu gelangen.

Nachrichten-Referenz – Usermanager

GB → UM {um, login,*Pid*, *Username*, *Password*} :
Ein Benutzer will sich anmelden.

UM → GB {um, welcome, *Cookie*} :
Diese Nachricht wird geschickt, um der Gornbar eine gültige Username/Passwort-Kombination anzuzeigen. Der Cookie dient als Session-Id, die sowohl auf der Servernode als auch auf der Clientnode gespeichert wird. Der Cookie ist für jede eingeloggte Gornbar eindeutig. Er wird später benutzt, um sicherzustellen, dass kritische Messages (also solche, die den Zustand auf dem Server ändern) nur von Clients akzeptiert werden, deren Identität bekannt und überprüft ist.

GB → UM {um, logout, *Cookie*, *Username*, *Message*} :
Der User will die Arbeitsgruppe verlassen. Die *Message* ist eine „Signoff"-Nachricht, die in Applikationen sinnvoll verarbeitet werden kann. Beispielsweise wird sie im Chat angezeigt, um den anderen Benutzern mitzuteilen, warum ein Benutzer den Chat verlassen hat. Der Zustand des Benutzers ändert sich auf `offline`. Alles was serverseitig mit dem Cookie assoziiert war, wird gelöscht.

C → UM {um, register, *Pid*, *Cookie*, *Username*} :
Ein Client registriert sich als Observer der Userliste. Der Usermanager schickt Änderungen in der Userliste, also z.B. Zustandsänderungen der User, an alle registrierten Observer.

C → UM {um, unregister, *Pid*, *Cookie*, *Username*} :
Client meldet sich als Observer ab. Keine weiteren Nachrichten des Usermanagers sollen an diese *Pid* geschickt werden.

C → UM {um, list, *Pid*} :
Fordert die Liste aller Benutzer an. Diese Nachricht wird von Clientapplikationen benutzt.

UM → C {um, list, *Userlist*} :
Die Liste aller Benutzer wird geschickt. Diese enthält `user`-Records.

GB → UM {um, shortlist, *Pid*} :
Fordert die Liste aller Benutzer an. Diese Nachricht wird von der Gornbar benutzt.

UM → GB {um, shortlist, *Userlist*} :
Die Liste aller Benutzer wird geschickt. Diese enthält Zwei-Tupel mit dem Benutzernamen und dem Zustand.

C → UM {um, info, *Pid*, *Username*} :
Ein Client fragt den öffentlichen Teil der Informationen über einen User ab.

UM → C {um, info, *Userinfo*} :
Der öffentlicher Teil der Userinformation wird an den Client geschickt. *Userinfo* enthält den `user`-Record für den Benutzer.

C → UM {um, pinfo, *Pid*, *Cookie*, *Username*} :
Ein Client fragt den privaten Teil der Userinformationen ab. Das ist nur für den eigenen User oder privilegierte Benutzer erlaubt. Andere Anfragen generieren eine `denied`-Message.

UM → C {um, pinfo, *Username*, *Userinfo*} :
Der Server sendet den privaten Teil der Userinformationen an den Client. *Userinfo* enthält den `puser`-Record für den Benutzer.

UM → C {um, denied, *Msg*} :
Der Usermanager hat einem Client aufgrund fehlender Privilegien den Zugriff auf Benutzerdaten verweigert. Die Begründung steht in dem String *Msg*.

C → UM {um, away, *Cookie*, *Username*, *Message*} :
Der User will sich `away` melden. Der Zustand muss entsprechend angepasst werden. Der Text kann in Applikationen (wie dem Chat) genutzt werden.

C → UM {um, back, *Cookie*, *Username*, *Message*} :
Der User ist nicht mehr `away`. Der Zustand ändert sich auf `online`. Die *Message* kann in Applikationen verarbeitet werden.

UM → C {um, status, *Newstate*, *Username*, *Message*} :
Der Zustand eines Users ändert sich. *Newstate* ist eins der Atome `away`, `back` oder `logout`. Die *Message* wurde vom Benutzer eingegeben und kann in Applikationen verwendet werden.

C → UM {um, setuser, *Cookie*, *Username*, *Puser*} :
Modifizieren von Usereigenschaften. Darf nur von einem privilegierten Benutzer oder von einem User auf seine eigenen Daten angewendet werden.

C → UM {um, deluser, *Cookie*, *Username*, *UsernameDst*} :
Der Benutzer *UsernameDst* soll gelöscht werden. Dies ist nur möglich, wenn der sendende Prozess von einem privilegierten Benutzer ausgeführt wird. Sollte sich ein privilegierter Benutzer selbst löschen, ist der Zustand des Gesamtsystems undefiniert.

UM → GB {um, watchdog, ping, *Pid*} :
Einerseits wird die Gornbar mit dieser Nachricht aufgefordert, eine Antwortnachricht zu schicken, andererseits kann sie feststellen, ob der Server noch läuft. Sollte diese Nachricht zu lange ausbleiben, geht die Gornbar von einer Nichterreichbarkeit des Servers aus und loggt sich aus. Die *Pid* ist die des Usermanagers, um die Beantwortung der Nachricht zu erleichtern.

GB → UM {um, watchdog, pong, *Pid*} :
Beantwortet die Watchdognachricht vom Usermanager. Sollte diese Nachricht zu lange ausbleiben, wird die Gornbar vom Usermanager für tot erklärt und ausgeloggt.

2.4.3 Der Applicationmanager

Der Applicationmanager dient der Verwaltung der Applikationen in Gorn. Dazu gehört das Verwalten und Ausliefern einer Liste der verfügbaren Applikationen und das Starten der Applikationen auf den Clientknoten.

Nachrichten-Referenz – Applicationmanager

G → AM {gorn, serverlist, *List*} :
Der Gorn-Prozess teilt dem Applicationmanager die Liste der gestarteten Server mit. Diese wird später beim Starten von Clients benötigt. Sie enthält Zwei-Tupel mit dem Servernamen und der PID.

C → AM {appman, list, *Pid*} :
Der Client *Pid* fordert eine Liste von verfügbaren Applikationen an.

AM → C {appman, list, *Applicationlist*} :
Der Applicationmanager teilt dem Client die verfügbaren Anwendungen mit. Die Liste enthält Tupel mit allen notwendigen Informationen, um die Knöpfe anzuzeigen und die Applikationen zu starten.

C → AM {appman, spawnstatics, *Client*, *Cookie*, *Username*} :
Wenn *Username* und *Cookie* korrekt sind, werden auf dem Client alle verfügbaren Clientstatics gestartet. Dies passiert beim Einloggen eines Benutzers auf der Gornbar.

AM → C {appman, spawnstatics, *Statics*} :
Der Applicationmanager teilt dem Client die Liste der gestarteten Statics mit.

AM → C {appman, spawnstatics, error, invalidcookie} :
Der Client hat einen falschen Cookie übermittelt.

C → AM {appman, start, *Client*, *Cookie*, *Username*, *App*, *Param*} :
Wenn *Username* und *Cookie* korrekt sind, wird auf dem Client die Applikation *App* mit den zusätzlichen Parametern *Param* gestartet.

AM → C {appman, start, *Pid*} :
Der Applicationmanager teilt dem Client die *Pid* des neuen Prozesses mit.

AM → C {appman, start, error, invalidcookie} :
Der Client hat einen falschen Cookie übermittelt.

C → AM {appman, logout, *Cookie*, *Username*} :
Der Client möchte ausgeloggt werden. Alle Prozesse des Clients bekommen eine `KILL`-Nachricht geschickt.

Bewertung

Der Applicationmanager hat nur Zugriff auf Prozesse, die er direkt gestartet hat. Sollte einer dieser Prozesse eigene Prozesse starten, kann der Applicationmanager diese nicht beenden.

Wenn ein Prozess terminiert, kümmert sich der Applicationmanager nicht um dessen Rückgabewert. Im normalen Betrieb ist das kein Problem, stürzt eine Applikation allerdings ab, wird die Fehlermeldung nicht sichtbar.

Wie bereits erwähnt, sucht der Applicationmanager nur beim Start nach neuen Applikationen. Dieses Caching macht es nötig, den GORN Server neu zu starten, wenn eine neue Applikation bereit gestellt werden soll.

2.4.4 Der DB-Server

Der Datenbankserver verwaltet die Daten, die in GORN langfristig gespeichert werden müssen. Er benutzt die in ERLANG enthaltene Datenbank Mnesia zur Datenablage. Der DB-Server kommuniziert nicht über Nachrichten mit den anderen Modulen, sondern stellt Funktionen zur Verfügung, die den Zugriff auf die Mnesia-Tabellen kapseln. Dadurch können nur serverseitige Module den DB-Server benutzen.

Zur Zeit werden drei Tabellen vom DB-Server gehalten, davon werden aber nur die ersten beiden tatsächlich auf der Platte abgelegt.

Die Daten der Benutzer werden in der Tabelle `dbuser` gespeichert.

Feld	Typ	Beschreibung
username	Atom	Username des Benutzers
fullname	String	Voller Name des Benutzers
privileged	Atom	`true`, wenn der Benutzer privilegiert ist, sonst `false`
passwd	String	Passwort

Die Mailapplikation benutzt das DB-Servermodul, um Emailnachrichten aufzubewahren. Dazu wird eine Tabelle von Nachrichten (`mailspool`) verwaltet, die die folgenden Felder enthält.

Feld	Typ	Beschreibung
msgid	Atom	Eindeutige Kennzeichnung einer Nachricht
user	Atom	Empfänger (als Benutzername)
from	Atom	Absender (als Benutzername)
destination	[Atom]	Alle Empfänger
time	`localtime`-Tupel	Zeit des Absendens
state	String	Zustand (N=neu, D=gelöscht, _=gelesen)
subject	String	Betreffzeile
body	String	Eigentliche Nachricht

Der aktuelle Zustand eingeloggter Benutzer wird in einer flüchtigen Mnesia-Tabelle `user_state` gehalten, um diese Informationen einfacher ERLANG-global verfügbar zu machen. Diese Tabelle wird bei jedem Starten von GORN angelegt und beim Ein-/Ausloggen aktualisiert.

Feld	Typ	Beschreibung
username	Atom	Benutzername
cookie	Binary	Cookie des Benutzers
state	Atom	Aktueller Zustand (`online` oder `away`)

3 Applikationen

3.1 Die Gornbar

Der Funktionsumfang der Gornbar ist sehr gering, um das Modul, das auf den Clientrechnern installiert werden muss, klein zu halten. Die Benutzung der Gornbar wird in Anhang A erläutert, deshalb soll hier nur auf die Nachrichten eingegangen werden, die die Gornbar versteht.

Nachrichten-Referenz – Gornbar

C → **GB** {gornbar, selectedusers, *Pid*} :
Eine Applikation erfragt die Liste der momentan in der Userliste ausgewählten User. Diese Nachricht wird beispielsweise vom Talk-Client geschickt, um einen Talk-Partner zu ermitteln.

GB → **C** {gornbar, selectedusers, *Userlist*} :
Die Gornbar meldet die Usernamen der ausgewählten Benutzer.

C → **GB** {log, *Message*} :
Message muss ein String sein, der dann von der Gornbar im Logfenster ausgegeben wird.

Weiterhin versteht die Gornbar noch diverse Nachrichten des User- und des Applicationmanagers. Sie registriert sich beim Usermanager, um Statusänderungen der anderen Benutzer anzeigen zu können. Weiterhin benutzt sie den Applicationmanager, um die Liste der verfügbaren Applikationen auszulesen sowie Applikationen zu starten.

Bewertung

Wenn ein Benutzer seinen Zustand ändert, also sich beispielsweise ein- oder ausloggt, sieht das Protokoll die Übergabe von Nachrichten vor. Diese Nachrichten sind momentan fest vorgegeben. Die Gornbar könnte diese vom Benutzer erfragen. Es wäre auch möglich, diese Nachrichten dem Benutzerprofil zuzuordnen, so dass jeder Nutzer sie nur einmal eingeben muss.

Wenn weitere Applikationen implementiert werden, muss die grafische Oberfläche der Gornbar angepasst werden. Das Raster, in dem die Applikationsknöpfe angezeigt werden, ist im Moment fest, so dass nur eine begrenzte Anzahl von Knöpfen angezeigt werden kann.

3.2 Das Mailsystem

Das Mailsystem dient dem Austausch elektronischer Nachrichten. Jeder Benutzer hat ein so genanntes *Mailspool*, in dem alle Nachrichten dieses Benutzers innerhalb der Datenbank gespeichert werden. Nachrichten werden mit dem Mailclient verfasst und abgeschickt. Natürlich können Nachrichten auch an Benutzer geschickt werden, die gerade nicht eingeloggt sind. Die Nachrichten können ebenfalls mit dem Mailclient gelesen werden. Neue Nachrichten werden als neu markiert, damit der Benutzer die ungelesenen Nachrichten sofort erkennen kann. Nachrichten können beantwortet (*reply*), weitergeschickt (*forward*) und als gelöscht markiert werden (*delete*). Als gelöscht markierte Nachrichten können dann endgültig aus dem Mailspool entfernt werden (*expunge*).

Das Zustellen und Ausliefern der Mails übernimmt die Serverside der Mailapplikation, das Lesen und Schreiben der Clientteil. Weiterhin gibt es noch ein Clientstaticmodul, das sich bei der Serverside registriert und den Benutzer mit dem Nachrichtenfenster der Gornbar über eingegangene Mail informiert. Beim Einloggen überprüft der Clientstaticteil, ob der Benutzer ungelesene Nachrichten besitzt und zeigt gegebenenfalls eine entsprechende Nachricht im Logfenster der Gornbar an. Über den selben Registrierungsmechanismus sorgt auch der Mailclient dafür, dass die Nachrichtenliste immer aktuell ist, d.h. wenn eine Nachricht eintrifft und der Mail-Client läuft gerade, wird eine neue Liste der Mails angefordert. Dieser Informationsaustausch über neue Mails ist in Abbildung 4 als *biff* bezeichnet.

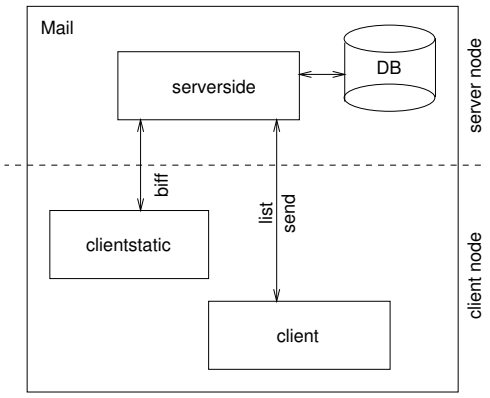


Abbildung 4: Mail-Applikation

Nachrichten-Referenz – Mail

CS/C → S {mail, register, *Pid*, *Cookie*, *Username*} :
Ein Clientstatic oder Client meldet sich beim Mail-Server an, um „You have new Mail“™ Benachrichtigungen zu erhalten.

CS/C → S {mail, unregister, *Pid*, *Cookie*, *Username*} :
Ein Clientstatic oder Client meldet sich wieder ab.

S → CS/C {mail, you_have_new_mail_tm, *Username_src*} :
Eine Mail von *Username_src* ist angekommen.

C → S {mail, list, *Pid*, *Cookie*, *Username*} :
Fordert alle Mails aus der Mailbox von Benutzer *Username* an.

S → C {mail, list, *Listofmessages*} :
Liefert alle Messages aus der Mailbox in einer Liste von `message`-Records.

C → S {mail, send, *Cookie*, *Username*, *Message*} :
Schickt eine Mail vom Benutzer *Username* an den/die Empfänger, der/die in der *Message* (`sendmessage`-Record) angegeben ist/sind. Ungültige Nachrichten erzeugen eine Bounce-Mail.

C → S {mail, setstate, *Cookie*, *Username*, *MsgId*, *State*} :
Setzt den Status der angegebenen *MsgId* auf *State*.

C → S {mail, expunge, *Cookie*, *Username*} :
Entfernt alle Messages aus der Mailbox des Benutzers *Username*, deren Status „D“ (deleted) ist.

Bewertung

Die Liste der Email-Nachrichten sollte der besseren Übersichtlichkeit halber nach dem Eingangsdatum sortiert sein, dies ist aber nicht implementiert. Daher werden die Nachrichten in der eher zufälligen Reihenfolge angezeigt, in der sie in der Datenbank gespeichert sind.

Zur Verbesserung der Benutzerfreundlichkeit sollte noch eine „Undelete“ Funktion hinzugefügt werden, um die Löschmarkierung zurück zu setzen.

3.3 Das Chatsystem

Über das Chatsystem können mehrere Benutzer miteinander kommunizieren. Die Serverside verwaltet genau einen Chatkanal, in dem alle Benutzer sprechen können, die die Chatapplikation gestartet haben. Neben dem normalen öffentlichen Sprechen kann ein Benutzer auch im Chat „agieren“, indem er dem Text ein `/me` voranstellt. In der Benutzerliste der Clientapplikation werden alle Benutzer aufgelistet, die momentan im Chat sind. Wenn ein Benutzer seinen Status wechselt, beispielweise „away“ geht, wird diese Tatsache im Chatfenster ausgegeben. Wenn ein Benutzer eine Zeile Text eingegeben hat, wird diese an den Server geschickt, welcher sie an alle angeschlossenen Chat-Clients weiterleitet. Es existiert kein Clientstatic-Teil im Chatsystem. Abbildung 5 stellt die beteiligten Komponenten dar.

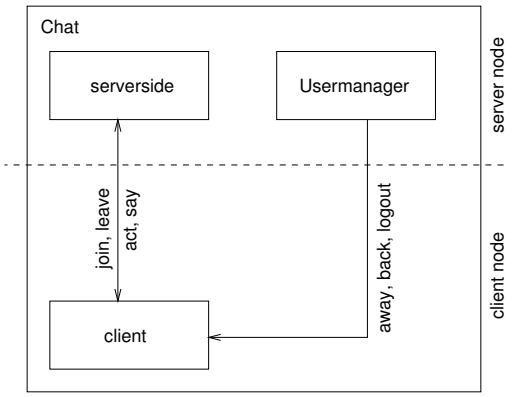


Abbildung 5: Chat-Applikation

Als kleiner Bonus wird im Chatfenster ein „Info“-Knopf angezeigt, der nach dem Markieren eines oder mehrerer Benutzer in der Benutzerliste Informationen zu diesen Benutzern im Ausgabefenster anzeigt.

Die Verwaltung der Benutzer im Chat erfordert, dass der Server immer weiß, welche Benutzer sich gerade im Chat befinden. Dazu registrieren sich die Chatclients beim Starten beim Chatserver mit einer *join*-Nachricht. Die Antwort enthält eine Liste aller Benutzer im Chat. Sollte ein weiterer Benutzer den Chat betreten oder ihn verlassen, werden alle registrierten Clients benachrichtigt. Auf diese Weise verwaltet jeder Client seine eigene Liste von Chatpartnern.

Zusätzlich registrieren die Clients sich beim Usermanager, um den Status der anderen Benutzer zu überwachen. Wenn ein Teilnehmer sich ein- oder ausloggt bzw. „away“ geht, wird eine entsprechende Meldung im Chatfenster ausgegeben.

Nachrichten-Referenz – Chat

C → **S** {chat, join, *Pid*, *Cookie*, *Username*} :
Ein Client möchte am Chat teilhaben. Der Server prüft *Cookie* und *Username* und fügt *Pid* und *Username* seiner Clientliste hinzu.

S → **C** {chat, welcome, *Listofusers*} :
Der Server hat den Client in den Chat aufgenommen und übermittelt die Liste der teilhabenden Benutzer.

S → **C** {chat, join, *Username*} :
Ein weiterer Benutzer betritt den Chat. Der *Username* wird in die angezeigte Userliste aufgenommen.

C → **S** {chat, leave, *Cookie*, *Username*, *Message*} :
Der Client möchte den Chat mit der gegebenen Signoff-Message verlassen. Der Server teilt das allen angeschlossenen Teilnehmern mit.

S → **C** {chat, leave, *Username*, *Message*} :
Ein Benutzer verlässt den Chat. Der *Username* wird aus der Userliste zu entfernt. Die *Message* (Signoff-Message des Users) wird im Chatfenster ausgegeben („username has left the chat 'message'").

C → **S** {chat, say, *Cookie*, *Username*, *Text*} :
Der User *Username* spricht eine Zeile öffentlich in den Chat.

S → **C** {chat, say, *Username*, *Text*} :
Ein User spricht öffentlich. Der einzeilige *Text* wird im Chatfenster ausgegeben.

C → **S** {chat, act, *Cookie*, *Username*, *Text*} :
Der Benutzer *Username* „agiert" im Chat (`/me` Effekt).

S → **C** {chat, act, *Username*, *Text*} :
Ein User agiert im Chat. Der *Text* wird entsprechend formatiert im Chatfenster ausgegeben (`/me` Effekt).

Bewertung

Der Chat ließe sich durch mehrfaches Instanziieren des Chatservers so erweitern, dass mehrere Chatkanäle verfügbar wären. Der Client müsste so erweitert werden, dass er den Server des gewünschten Chatkanals unter den Verfügbaren auswählt und sich bei diesem anmeldet.

3.4 Das Talksystem

Im Gegensatz zum Chat können beim Talk nur genau zwei Benutzer miteinander kommunizieren. Ein weiterer Unterschied ist, dass jedem Benutzer im Talkclient ein Fenster für seinen eingegebenen Text zur Verfügung steht und der Benutzer explizit eine Verbindung mit einem anderen Benutzer aufnehmen kann. Der andere Benutzer erhält dann ein Benachrichtigungsfenster, in dem er die Anfrage annehmen oder ablehnen kann. Nimmt er an, wird die Talkapplikation gestartet.

Die Nachfrage beim Benutzer übernimmt die Clienstatickomponente des Talksystems, nachdem sie von der Serverside eine Talk-Anfrage des anderen Benutzers erhalten hat. Die Serverside beschränkt sich auf die Vermittlung von Talk-Anfragen, die eigentliche Kommunikation zwischen den Benutzern findet ohne Umweg zwischen den Talkclients statt.

Die Kommunikation zwischen dem Clientstatic und dem Talkserver wird dadurch ermöglicht, dass der Clientstatic sich, wenn er gestartet wird (also wenn der Benutzer sich einloggt), beim Talkserver registriert. Sobald nun ein Talkclient eine Anfrage an den Server schickt, ermittelt dieser anhand des übergebenen Benutzernamens den zum Partner gehörenden Clientstatic und leitet die Anfrage an diesen weiter. Läuft kein Clientstatic für diesen Benutzer, wird die Anfrage vom Server abgelehnt.

Sollte die Anfrage vom Gegenüber abgelehnt werden, verschickt der Clientstatic die Ablehnung an den initiierenden Talkclient. Wird die Anfrage angenommen, startet der Clientstatic den Talkclient, indem er dem Applicationmanager eine Nachricht zum Starten der Talkapplikation schickt. Sobald vom Applicationmanager eine positive Rückmeldung kommt (die die PID des gestarteten Talkclients enthält), wird dem Talkpartner die PID übermittelt und die Verbindung besteht.

Um das automatische Starten des Talkclients bei Verbindungsannahme zu ermöglichen, kann der Client in zwei Modi starten. Wenn er normal (also aus der Gornbar) gestartet wird, erfragt er bei der Gornbar die in der Userliste markierten Benutzer und verschickt eine Talkanfrage, wenn exakt ein Benutzer markiert ist. Im anderen Modus, also wenn der Client vom Clientstatic gestartet wird, bekommt er einen zusätzlichen Parameter übergeben, der den Benutzernamen und die PID des Gegenübers enthält.

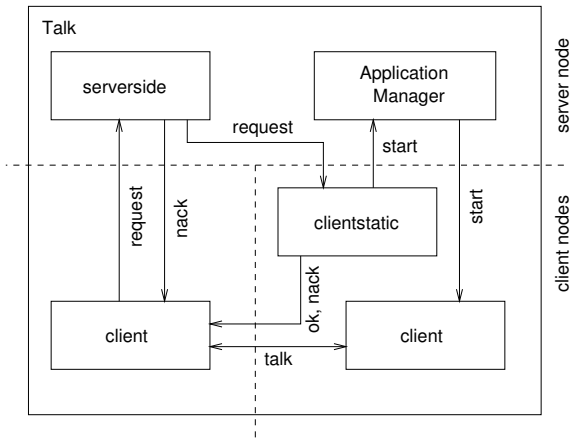


Abbildung 6: Talk-Applikation

Nachrichten-Referenz – Talk

CS → **S** {talk, register, *Pid*, *Cookie*, *Username*} :
Ein Clientstatic registriert sich für den Benutzer *Username* beim Server, damit dieser Talkanfragen weiterleiten kann.

CS → **S** {talk, unregister, *Pid*, *Cookie*, *Username*} :
Entfernt den Clienstatic für *Pid* aus der Liste des Talkservers.

C → **S** {talk, request, *Username_dst*, *Pid*, *Cookie*, *Username_src*} :
Ein Client (*Username_src*) möchte mit dem gegebenen User (*Username_dst*) talken. Die *Pid* ist der Talkprozess. Die Serverside überprüft die Anfrage und leitet sie weiter.

S → **CS** {talk, request, *Username_src*, *Pid*} :
Der Server vermittelt eine Talkanfrage des Users *Username_src* vom Prozess *Pid*.

C → **C** {talk, ok, *Pid*} :
Ein Client hat die Talkanfrage eines anderen Clients angenommen. Die *Pid* ist der eigene Prozess. Diese *Pid* und der Initiator können sich jetzt unterhalten.

C → **C** {talk, nack, *Username*} :
User lehnt ab, mit dem Client zu reden (direkte Verbindung).

S → **C** {talk, nack, *Username*} :
User ist momentan nicht online, deshalb lehnt der Server die Anfrage ab.

C → **C** {talk, insert, *Text*} :
Austausch eines getippten Zeichens.

C → **C** {talk, disconnect} :
Beenden einer Talk-Verbindung.

Bewertung

Eine Einschränkung des ERLANG-Grafiksystems hat sich bei der Entwicklung der Talkapplikation gezeigt. Wenn Text mit der Maus kopiert und in das Eingabefenster eingefügt wird, kann dieser nicht an den Talkpartner übermittelt werden. Der Grund dafür ist, dass es keine dokumentierte Möglichkeit gibt, solche „Copy & Paste“-Events abzufragen.

3.5 Die Verwaltungsapplikation

Das Admin-Tool dient dazu, Benutzer-Accounts anzulegen, zu bearbeiten und zu löschen. Privilegierte User können die Daten aller anderen User ändern, unprivilegierte User dürfen nur ihre eigenen Daten bearbeiten. Es gibt keinen Clientstatic-Teil. Der Client kommuniziert direkt mit dem Usermanager, um Userdaten abzufragen bzw. sie eintragen zu lassen.

Die Daten eines Benutzers umfassen seinen Usernamen (der nicht geändert werden kann), einen vollen Namen, sein Passwort und die Angabe, ob der User privilegiert ist oder nicht.

Die Admin-Applikation hat keine eigenen Nachrichten, es werden ausschließlich die Nachrichten des Usermanagers zur Abfrage und Modifikation der User-Daten benutzt.

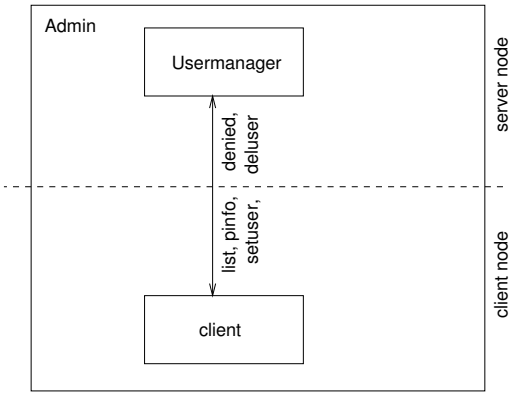


Abbildung 7: Verwaltungsapplikation

Bewertung

Zur Zeit beschränkt sich die Verwaltungsapplikation auf die Verwaltung der Benutzerdaten. Eine sinnvolle Erweiterung würde die Verwaltung der Applikationen darstellen, beispielsweise die Möglichkeit, Programmdateien auf dem Server zu installieren.

3.6 Die Spiele

Die Funktionalität, die allen Spielen gemeinsam ist, wurde in einen Server ausgelagert, der entsprechend den anderen Applikationsservern vom GORN-Prozess auf dem Serverknoten gestartet wird. Dieser Server wird in Abschnitt 3.6.1 näher beschrieben.

Weiterhin bestehen die Spiele aus einem spielspezifischen Serverteil, der den Spielverlauf koordiniert und berechnet, sowie aus einem Clientteil, der für die Darstellung der Spiele auf dem Bildschirm zuständig ist. Der Serverteil läuft auf dem Clientknoten des Benutzers, der ein Spiel startet. Die Clientteile laufen auf den Knoten der teilnehmenden Spieler. Die Kommunikation zwischen Server- und Clientprozessen verläuft direkt zwischen den Knoten der Spieler und dem Spielserver (siehe Abbildung 8). Server- und Clientteil eines Spiels sind in einem einzigen Modul implementiert.

Die Rolle der spielspezifischen Server unterscheidet sich von Spiel zu Spiel. Der Server kann zentral den Spielfortschritt berechnen und damit den Hauptteil der Arbeit übernehmen, so dass die Clients lediglich der Ein-/Ausgabe dienen. Es ist ebenfalls möglich, dass die Clients ihren Spielverlauf autonom berechnen und der Server lediglich zur Synchronisation und Koordinierung benutzt wird.

Die serverzentrierten Spiele Pong und Tron verwenden zur Synchronisation des Spielverlaufs einen *Gametick*-Mechanismus. Dabei wird der Spielfortschritt in diskrete Zeitabschnitte unterteilt. Beginn und Dauer eines Abschnitts wird vom Server vorgegeben.

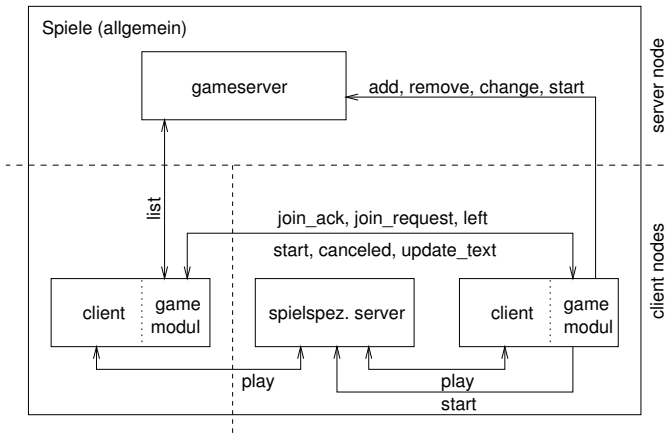


Abbildung 8: Allgemeines Kommunikationsschema der Spiele

3.6.1 Der Gameserver

Der Gameserver ist die zentrale Instanz zur Verwaltung von Spielen. Er hält eine Liste von Spielen, die auf Teilnehmer warten. Zusätzlich stellt er sicher, dass Clients authentifiziert sind, da er die Liste der verfügbaren Spiele nur ausliefert, wenn der Benutzer ordnungsgemäß angemeldet ist. Die Aushandlung von Spieleparametern wird vom `game`-Modul übernommen (Abschnitt 3.6.2).

Nachrichten-Referenz – Gameserver

C → GS {game, add, *Pid*, *Cookie*, *Username*, *Gamename*, *Missing*} :
Der Client fordert den Gameserver auf, das Spiel *Pid* vom Typ *Gamename*, gestartet von *Username* mit *Missing* fehlenden Spielern der Liste von Spielen hinzuzufügen.

C → GS {game, remove, *Pid*, *Cookie*, *Username*} :
Der Client fordert den Gameserver auf, das Spiel *Pid* aus seiner Spieleliste zu entfernen.

C → GS {game, change, *Pid*, *Cookie*, *Username*, *Missing*} :
Der Client meldet dem Gameserver eine Änderung der Anzahl fehlender Spieler für das Spiel *Pid*.

C → GS {game, start, *Pid*, *Cookie*, *Username*} :
Der Client meldet dem Gameserver, dass das Spiel *Pid* gestartet wurde. Der Gameserver entfernt es daraufhin aus der Liste wartender Spiele.

C → GS {game, list, *Pid*, *Cookie*, *Username*, *Gamename*} :
Der Client bittet den Gameserver um eine Liste wartender Spiele.

GS → C {game, list, *Games*} :
Der Gameserver schickt dem `game`-Modul eine Liste von verfügbaren Spielen. Die Listenelemente sind Tupel mit folgendem Aufbau: {*Spieltyp*, *Username*, *ServerPid*, *Status* (`waiting` oder `running`), *Missing*}

3.6.2 Das `game`-Modul

Das `game`-Modul ist ein generisches Modul, um die Synchronisation zwischen einem Spielserver und mehreren Clients zu ermöglichen. Es stellt die gesamte Funktionalität zum Aushandeln eines Mehrspielerspiels zur Verfügung.
Spiele, die das `game`-Modul benutzen, müssen die folgenden zwei Funktionen bereitstellen:

`gameconfig(`*`Param, Oldconfig`*`)`: Diese Funktion wird vom `game`-Modul aufgerufen um die Parameter für das Spiel einzustellen. Die Funktion wird mit den Startparametern der Gorn Applikation und einer Liste der Optionen aufgerufen (diese Liste ist beim ersten Aufruf leer). Als Rückgabewert erwartet das `game`-Modul ein Zwei-Tupel mit einem darstellbaren Text, der die Einstellungen beschreibt, und eine Liste mit den neuen Einstellungen oder `false`, falls der Benutzer *Cancel* angewählt hat.

`server_loop(`*`Params, Config, Clientlist`*`)`: Diese Funktion muss die Serverfunktionalität des Spiels implementieren. Sie wird mit den Startparametern, den Spieleinstellungen und einer Liste der Clients aufgerufen. Diese Liste enthält Usernamen und PIDs der Teilnehmer.

Um das Aushandeln eines Spiels zu starten, müssen die Spiele `game:connect(`*Modulname*`,` *Parameterliste*) aufrufen. Die Funktion kehrt entweder mit dem Pid des Spielservers zurück oder mit `false`, falls der Benutzer den Vorgang abgebrochen hat.

Die Dialoge des `game`-Modul

Der Join-Dialog: In diesem Dialogfeld wird die Liste der verfügbaren Spiele alle fünf Sekunden vom Gameserver abgeholt und angezeigt. Der Benutzer hat die Möglichkeit einem Spiel beizutreten oder ein neues zu erstellen. Wenn er ein neues Spiel erstellt, wird an den Gameserver eine {`game, add, ...`}-Nachricht geschickt.

Der Create-Dialog: In diesem Dialog werden die Einstellungen für ein Spiel verwaltet. Der Benutzer kann Spieleinstellungen ändern, die maximale Anzahl der Spieler festlegen, bestimmte Spieler von der Teilnahme ausschließen und das Spiel starten.

Der Start-Dialog: In diesem Dialogfeld wartet der Spieler auf den Start des Spiels bzw. kann noch aus dem Spiel aussteigen. Der Start wird vom Initiator in dessen Create-Dialog ausgelöst.

Die Nachrichten des `game`-Moduls

Nachrichten-Referenz – Join-Dialog

C → **C** {game, join_ack, *Gameserver*, *Text*} :
Der Create-Dialog *Gameserver* bestätigt die Teilnahme an dem Spiel. Der Join-Dialog wird geschlossen und der Start-Dialog mit *Text* angezeigt. Der *Text* enthält einen zweizeiligen String, der die Konfiguration des Spiels in einer lesbaren Form enthält.

Nachrichten-Referenz – Create-Dialog

C → **C** {game, join_request, *Username*, *Pid*} :
Der Join-Dialog *Pid* mit Benutzer *Username* möchte am Spiel teilnehmen.

C → **C** {game, left, *Username*, *Pid*} :
Der Start-Dialog *Pid* mit Benutzer *Username* ist aus dem Spiel ausgestiegen.

Nachrichten-Referenz – Start-Dialog

C → **C** {game, start, *Server*} :
Der Create-Dialog hat das Spiel gestartet, der Server für dieses Spiel hat die PID *Server*.

C → **C** {game, canceled} :
Das Spiel wurde abgebrochen bevor es gestartet wurde.

C → **C** {game, update_text, *Text*} :
Der Status des Spiels hat sich geändert, der dargestellte Text soll auf *Text* geändert werden.

3.6.3 GORNtris

GORNtris ist ein Tetris™ mit Multiplayer-Funktionalität. Ziel des Spieles ist nicht, wie beim Einspieler-Tetris™, so viele Zeilen wie möglich zu entfernen, sondern als letzter Spieler übrig zu bleiben. Bei GORNtris erhalten alle Spieler die gleichen Steine in der selben Reihenfolge. Zu diesem Zweck werden die Steine vom Server generiert und dort in einer Liste verwaltet.

Das Spiel ist verloren, wenn GORNtris beim Erzeugen eines neuen Steins feststellt, dass dieser nicht mehr bewegt werden kann, also das Spielfeld bis oben mit Steinen gefüllt ist. Um das Spiel zu gewinnen muss der Spieler einerseits seine Steine so platzieren, dass er weiterhin handlungsfähig bleibt, andererseits muss er seinen Mitspielern dies erschweren. Zu diesem Zweck kann er, durch das gleichzeitige Entfernen mehrerer Zeilen, seinen Mitspielern von unten Zeilen zum Spielfeld hinzufügen. Diese Strafzeilen enthalten an einer zufällig ausgewählten Stelle eine Lücke, damit sie nicht sofort wieder verschwinden. Für das Einfügen der Strafzeilen sind zwei verschiedene Modi implementiert: Der erste Modus fügt die Zeilen jederzeit ein (*always*), der zweite Modus akkumuliert die einzufügenden Zeilen bis das nächste mal ein Block abgesetzt wird (*drop*).

GORNtris verwaltet alle Informationen das Spielfeld betreffend in den *rectangle*-Objekten des Canvas. Das bedeutet, dass es keine weitere Datenstruktur mit den Spielfeldinformationen gibt. GORNtris erkennt einen Block des Spielfeldes als *frei*, wenn dieser die Farbe schwarz hat. Da die Wände des Spielfeldes grau sind, ist es nicht möglich einen Stein in die Wand zu schieben.

Die Steine von GORNtris bestehen jeweils aus vier Blöcken. Es gibt sieben unterschiedliche Arten von Blöcken, die ein, zwei oder vier Rotationszustände haben können. Die sieben unterschiedlichen Arten heißen `quad, long, four, stool, el, elinv, eh`. Sie sind in Abbildung 9 dargestellt. Jeder Stein hat einen Dreh- und Anfasspunkt, der zur Berechnung der restlichen drei Blöcke herangezogen wird.

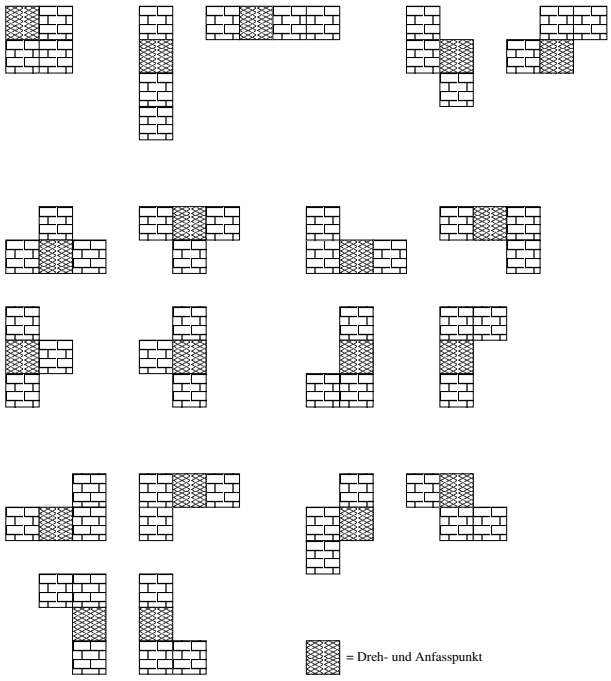


Abbildung 9: Die verschiedenen Steine in ihren unterschiedlichen Rotationszuständen.

Zusätzlich zu den Spielinformationen auf dem Client werden auf dem Server Informationen über die aktuelle Spielgeschwindigkeit, die Anzahl der entfernten Zeilen und die Liste der aktiven/ausgeschiedenen Spieler verwaltet.

Nachrichten-Referenz – GORN**tris**

GS → **C** {tetris, config, *Gameconfig*, *ClientList*} :
Diese Nachricht wird vom Spieleserver nach dem Drücken des *Start*-Knopfes an alle Clients geschickt, um ihnen die Konfiguration für das Spiel mitzuteilen. *Gameconfig* enthält die Größe des Spielfeldes, die Fallgeschwindigkeit, den ersten Stein und den Einfügemodus. *ClientList* enthält Usernamen und PIDs der Mitspieler.

C → **GS** {tetris, ready, *Pid*} :
Hiermit bestätigt ein Client den Empfang der Konfiguration und das erfolgreiche Erzeugen des Spielfeldes.

GS → **C** {tetris, start} :
Alle Clients haben den Empfang der Konfiguration bestätigt, und das Spiel wird gestartet.

C → **GS** {tetris, nextstone, *Pid*} :
Der Client *Pid* fordert den nächsten Stein an.

GS → **C** {tetris, nextstone, *Stone*} :
Mit dieser Nachricht wird dem Client der nächste zu verwendende Stein mitgeteilt. Diese Nachricht wird vom Server nach dem Empfang der obigen Nachricht erzeugt.

C → **GS** {tetris, removed, *Pid*, *NumLines*} :
Der Client *Pid* meldet, dass bei ihm *NumLines* Zeilen entfernt wurden.

GS → **C** {tetris, insert, *NumLines*} :
Mit dieser Nachricht wird der Client aufgefordert, *NumLines* Zeilen von unten dem Spielfeld hinzuzufügen. Je nach Einfügemodus wird entweder sofort eingefügt, oder der Wert zwischengespeichert und das Einfügen nach dem Absetzen des nächsten Steins vorgenommen.

C → **GS** {tetris, height, *Pid*, *Height*} :
Der Client *Pid* meldet, dass der höchste Stein im Spielfeld auf der Höhe *Height* ist.

GS → **C** {tetris, height, *Pid*, *Height*} :
Mit dieser Nachricht wird die Höhenanzeige für den Client *Pid* aktualisiert.

GS → **C** {tetris, speedup, *NewSpeed*} :
Mit dieser Nachricht setzt der Server eine neue Fallgschwindigkeit der Steine auf dem Client. Diese wurde zentral für alle Spieler auf dem Server berechnet.

C → **GS** {tetris, dead, *Pid*} :
Ein Spieler meldet sich tot, weil sein Spielfeld voll geworden ist oder er das Spiel vorzeitig verlassen hat.

GS → **C** {tetris, dead, *Pid*} :
Diese Nachricht teilt allen Clients das Ausscheiden eines Spielers mit. Die Höhenanzeige für diesen Spieler wird bis zum oberen Fensterrand gezogen und rot eingefärbt.

GS → **C** {tetris, winner, *Pid*} :
Mit dieser Nachricht wird der Client *Pid* zum Sieger erklärt.

Bewertung

Der Einfügemodus *always* ist nicht optimal gelöst. Es kann beim Einfügen von Zeilen passieren, dass der Stein in das bereits vorhandene Spielfeld hineingemischt wird, da während des Hinzufügens der Zeilen keine Kollisionserkennung stattfindet. Außerdem ist der Stein während des Einfügens ausgeblendet und nicht steuerbar, was das Spiel zusätzlich verkompliziert.

Im Einspielermodus gibt es kein Punktesystem oder ähnliche Bewertungskriterien. Ein Kriterium zur Punktevergabe könnte die Anzahl der abgeräumten Zeilen oder die Dauer bis zum Ausscheiden sein.

Das Rauf-/Runterbewegen des Spielfeldes ist sehr langsam, da die Informationen jedesmal für das ganze Spielfeld zeilenweise kopiert werden.

3.6.4 Pong

Pong ist ein Spiel für beliebig viele Spieler. Bei dem Spielfeld handelt es sich um ein Polygon, in dem sich ein Ball mit konstanter Geschwindigkeit bewegt. Jede zweite Seite des Polygons ist durchlässig und einem Spieler zugeordnet. Diese werden als *Homezones* bezeichnet. Die anderen Seiten sind fest. Aufgabe der Spieler ist es, den Ball nicht aus dem Polygon fliegen zu lassen. Dazu hat jeder Spieler ein Paddle, welches auf der Homezone bewegt werden kann. Eine Runde wird verloren, wenn der Ball die eigene Homezone passiert, ohne vom Paddle reflektiert zu werden.

Jeder Spieler sieht die eigene Seite am unteren Spielfeldrand und kann die angezeigte Größe des Spielfeldes frei wählen. Daher ist es nötig, in jedem Client die Universumskoordinaten durch Drehen, Skalieren und Verschieben dem Koordinatensystem des Anzeigecanvas anzupassen. Das Drehen ist nötig, damit jeder Spieler seine eigene Homezone am unteren Rand des Spielfeldes sieht. Durch den Skalierungsschritt ist die Größe des Fensters beliebig veränderbar. Die Verschiebung (Translation) ist nötig, da das Canvas den Koordinatenursprung links oben hat und nur positive Koordinaten sichtbar sind. Die Berechnung findet aber in einem normalen karthesischen Koordinatensystem statt, und sichtbare Werte sind sowohl negativ als auch positiv. Das Spielfeldpolygon hat Diagonalen, die 1000 Meter lang sind. Die Einheit ist willkürlich gewählt, erlaubt aber dem Benutzer eine gewisse Vorstellung der Geschwindigkeiten. Diese Vorstellung ist beim Einstellen der Spielparameter von Vorteil.

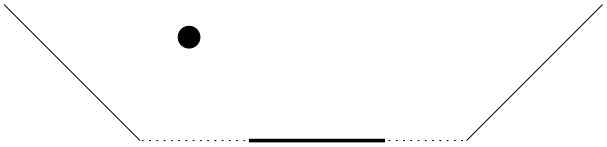


Abbildung 10: Sicht jedes Spielers auf die eigene Homezone. Die Wände links und rechts sind fest. Die Winkel und die Länge der Homezone ändern sich mit der Anzahl der Spieler. Zur Anzeige wird bei jedem Spieler das Universumskoordinatensystem auf das Koordinatensystem des Canvas transformiert.

Im Ponguniversum befinden sich der Ball und eine Anzahl von Strecken, die die Hindernisse darstellen. Der Ball hat keine Ausdehnung und hat zu jedem Zeitpunkt eine Position und einen Bewegungsvektor. Die Position ist ein einfaches Tupel aus der X- und der Y-Koordinate. Der Bewegungsvektor besteht aus einem Vektor der Länge 1, der die Richtung vorgibt, und einer Längenkomponente. Diese Aufteilung ist bei der Berechnung vorteilhaft, da Richtung und Länge teilweise voneinander getrennt benutzt werden. Außerdem bleibt die Genauigkeit erhalten, wenn ein Vektor sehr kurz werden sollte.

Ein Spielschritt besteht daraus, die Bewegungen der Spieler entgegenzunehmen und den Ball weiterzubewegen. Dabei muss der Ball an Hindernissen reflektiert und das Überschreiten einer Homezone angezeigt werden. Um Kollisionen mit einer Strecke zu berechnen, kommt das in Abbildung 11 gezeigte Schema zum Einsatz. Die Bezeichnungen lehnen sich an die Variablenbezeichnungen in den entsprechenden Funktionen an. Die Idee ist, den Schnittpunkt mit einer Strecke sowie den neuen Bewegungsvektor von diesem Punkt aus zu berechnen. Dazu werden die Geraden der Strecke und der Flugbahn konstruiert und der Schnittpunkt bestimmt. Liegt der Schnittpunkt auf der Strecke, ist eine Kollision aufgetreten. Aus der Länge des Bewegungsvektors und dem Abstand des Balles zum Schnittpunkt wird bestimmt, wie weit der Ball von der Strecke wegbewegt werden muss. Die Richtung des neuen Vektors (der Ausfallswinkel) wird durch eine Konstruktion

von Geraden, Loten und Hilfspunkten bestimmt. Dieses Schema versagt allerdings, wenn der Ball senkrecht auf die Strecke trifft, genau auf der Strecke liegen bleibt oder schon auf der Strecke liegt. Diese Fälle werden in der Berechnung gesondert behandelt.

Ein weiterer Sonderfall tritt ein, wenn der Ball genau den Schnittpunkt von zwei oder mehr Strecken trifft. Dies wird explizit überprüft und eine andere Methode zur Bestimmung des Ausfallswinkels benutzt. Auch dort müssen die Fälle, dass der Ball genau im Schnittpunkt zum Liegen kommt bzw. dort schon liegt, besonders behandelt werden.

Die Bewegung des Balls ist nur vollständig, wenn in jedem Schritt die Länge des Bewegungsvektors „aufgebraucht“ wird. Dazu wird obiger Algorithmus rekursiv immer wieder auf die Strecken der Welt angewendet. Die Berechnungen sind dabei nicht auf Effizienz optimiert. Viele Zwischenergebnisse werden unnötig oft berechnet und nicht zwischengespeichert.

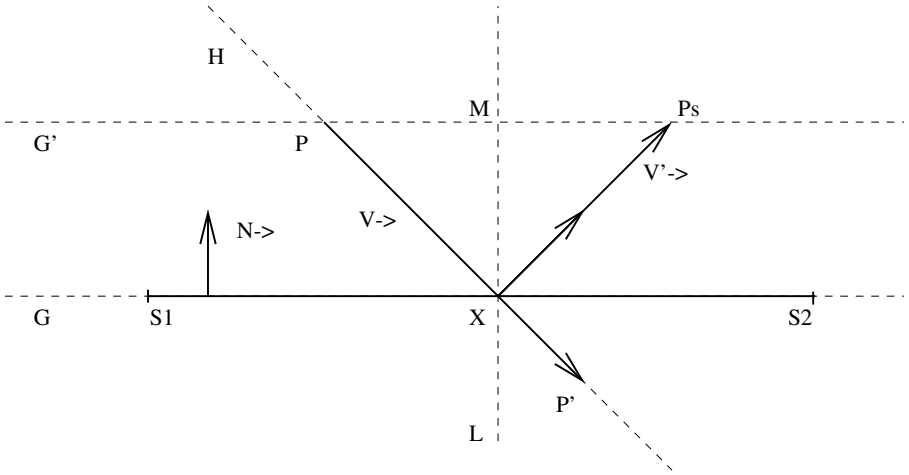


Abbildung 11: Konstruktion des Abprallens des Pongballs ohne Verwendung von Winkelfunktionen. Die Bezeichnungen entsprechen den Variablennamen in den Berechnungsfunktionen.

Um das Spiel etwas interessanter zu gestalten, ist die wirkliche Form des Paddles etwas anders als die angezeigte Form. Während eine einfache Strecke zu sehen ist, werden in Wirklichkeit drei Strecken berechnet: Die mittlere, die wie das angezeigte Paddle liegt, sowie links und rechts davon eine abgeschrägte Strecke. Somit ändert sich der Ausfallswinkel auf dem Paddle je nachdem, wo der Ball das Paddle trifft. Im Originalpong verändert sich der Ausfallswinkel zusätzlich mit der aktuellen Geschwindigkeit des Paddles. Dies ist aber nicht implementiert, da vom Paddle nur die Position, nicht aber die Bewegungsrichtung bekannt ist. Außerdem müsste die Berechungsmethode geändert werden, da im Moment kein Unterschied zwischen festen Wänden und Paddles gemacht wird bzw. keine Strecke einen Bewegungsvektor hat.

Der Initiator kann weiterhin automatische Spieler hinzufügen. Diese eigenständigen Prozesse bewegen ihre Paddles zufällig über ihre Homezone. So lassen sich größere Teilnehmerzahlen simulieren.

Nachrichten-Referenz – Pongspiel

PS → C {pong, welcome, *Numplayers*, *Playernum*, *Universe*, *Obstacles*, *Scorelist*} :
Diese Nachricht wird vom Server gesendet, sobald das Spiel gestartet werden soll und alle Mitspieler bekannt sind. *Numplayers* gibt die Anzahl der Mitspieler an, *Playernum* die eigene Spielernummer. Diese beiden Werte werden benötigt, um zu bestimmen, um welchen Winkel die Weltkoordinaten gedreht werden müssen, damit die eigene Homezone unten ist. Alle Spieler werden durch ihre Spielernummer identifizert. Das Argument *Universum* gibt die Ausdehnung des Universums an. Dieser Wert beträgt im Moment immer 1000. Alle

Berechungen im Universum finden also zwischen den X-/Y-Koordinaten von -500 bis 500 statt. *Obstacles* enthält eine Liste von Strecken, die fest im Universum liegen. Sie werden durch weiße Linien angezeigt. *Scorelist* enthält eine Assoziativliste mit Spielernamen (als Atom) und Punktesständen. Sobald er diese Nachricht erhalten hat, besitzt der Client genug Informationen, um das Anzeigefenster aufzubauen.

PS → **C** {pong, stuff, *Ballstate*, *Paddles*, *Steps*} :
Mit dieser Nachricht wird der Client aufgefordert, auch die beweglichen Elemente der Welt zu erzeugen. Der Client legt alle benötigten Linien und den Ball als grafische Elemente an. Später werden diese Elemente nur noch konfiguriert (und dadurch verschoben). Diese Nachricht wird gesendet, wenn eine neue Runde anfangen soll. *Ballstate* enthält Richtung und Position des Balles in einem gleichnamigen Record. *Paddles* ist eine Liste von Strecken, die die beweglichen Paddles darstellen. *Steps* ist die Anzahl der Schritte, die gebraucht werden, um das Paddle einmal komplett über die Homezone zu bewegen. Diese Information ist nur für die automatischen Spieler interessant, im normalen Client wird sie ignoriert.

PS → **C** {pong, info, *Text*} :
Diese Nachricht wird verwendet, um eine Zeile Text auf dem Client auszugeben. Dies sind Countdown und Spielfortschrittsanzeigen. Der Client wertet den Text nicht aus, sondern zeigt ihn nur an.

C → **PS** {pong, leave, *Playernum*} :
Diese Nachricht wird vom Client gesendet, wenn der Spieler das Spiel verlassen will. Verlassen kann der Spieler das Spiel, wenn er den "Close" Knopf betätigt oder das Fenster anderweitig geschlossen wird. Der Server identifiziert den Spieler anhand der Spielernummer *Playernum*. Damit die anderen Spieler weiterspielen können, wird das Paddle des Spielers über die ganze Breite der Homezone gezogen. Der Pongserver terminiert, wenn kein Spieler oder nur noch automatische Spieler vorhanden sind.

PS → **C** {pong, goaway} :
Der Server fordert die automatischen Spieler auf, das Spiel zu verlassen und ihre Prozesse zu beenden. Diese Nachricht wird nur an automatische Spieler gesendet.

PS → **C** {pong, scores, *List*} :
Mit dieser Nachricht kann der Server den Punktestand oder auch die Spielerliste aktualisieren. Das Format von *List* ist wiederum eine Assoziativliste von Spielernamen und Punktestand. Konzeptionell kann diese Liste jederzeit aktualisiert werden, praktisch passiert das aber immer zum Ende einer Runde oder wenn ein Spieler das Spiel verlässt.

PS → **C** {pong, update, *Ballstate*, *Paddles*} :
Der Server gibt ein Update der Elemente der Welt bekannt. *Ballstate* und *Paddles* enthalten wiederum die Ballposition und die Strecken, die die Paddles einnehmen (wie bei der `stuff`-Nachricht). Diese Nachricht kann prinzipiell jederzeit auftreten, wird aber nur benutzt, um den finalen Zustand der Welt anzuzeigen, wenn einer der Spieler den Ball fallenlässt, da dies nicht durch den `tick/tack`-Mechanismus (siehe unten) erledigt werden kann.

PS → **C** {pong, tick, *Ballstate*, *Paddles*} :
Dies ist die Nachricht, die das Spiel eigentlich vorantreibt. Der Server gibt den aktuellen Zustand der Welt an (wie oben) und fordert den Client auf, die Bewegungsrichtung des Paddles anzugeben. Der ganze Spielfluss ergibt sich aus dem Wechsel von `tick`- und `tack`-Nachrichten.

C → **PS** {pong, tack, *Playernum*, *Action*} :
Alle Clients antworten auf einen `tick` mit dieser Nachricht. *Action* beschreibt, was der Spieler mit seinem Paddle machen will: `left` um es nach links zu bewegen, `right` für rechts und `none` um es nicht zu bewegen. Der Server sammelt alle `tack`-Nachrichten auf, lässt die Länge eines Gameticks (kann in der Konfiguration zwichen 10 und 100 Millisekunden gewählt

werden) verstreichen und berechnet dann den Spielschritt, der mit der nächsten `tick`-Welle abgeschlossen wird. Wieviele Ticks pro Sekunde stattfinden, ist Abwägungssache. Je mehr es sind, desto flüssiger bewegt sich der Ball, desto höher ist aber auch die Netzwerklast.

Sobald der Pongserver alle Spieler mittels der `welcome`-Nachricht in das Spiel aufgenommen hat, beginnt er mit der Serverloop. Diese ist eine einfache tailrekursive Funktion, die einen großen `receive`-Block enthält. Der Spielserver hat einen Zustand: Das Spiel kann gerade im Gange sein, also hauptsächlich `tick/tack`-Nachrichten austauschen. Zwischen den Runden passiert das aber nicht, stattdessen wird ein Countdown durchgeführt, der allen Spielern erlaubt, sich auf die neue Runde vorzubereiten. Der Zustand des Spielservers (`gamestate`-Record) enthält aber keine explizite Zustandsvariable. Die Aktionen werden durch Nachrichten intern im Server koordiniert.

Nachrichten-Referenz – Pong, Serverinterne Koordination

PS → PS {pong, timefortick} :
Dies ist wohl die wichtigste Nachricht, da sie das Fortschreiten des Spiels forciert. Wenn der Pongserver diese Nachricht empfängt, berechnet er den nächsten Spielschritt und gibt einen `tick` aus. Ausgelöst wird diese Nachricht beim Empfangen der `tack`-Nachricht des letzten Clients, wenn also alle Spielzüge der Mitspieler bekannt sind. Allerdings wird `timefortick` nicht direkt ausgelöst, sondern mittels `timer:send_after`, welches diese Nachricht erst nach Ablaufen der Verzögerung zwischen zwei Ticks ausliefert. Sollte ein Berechnungschritt ergeben, dass ein Spieler den Ball fallengelassen hat, wird keine weitere Tickwelle ausgelöst, sondern nur noch die Anzeige der Clients aktualisiert und mit der folgenden Nachricht eine neue Runde begonnen.

PS → PS {pong, newround} :
Hiermit wird eine neue Runde begonnen, d.h. ein neuer Ballzustand bestimmt. Dazu wird der Ball zufällig in der Umgebung des Mittelpunktes platziert und auf eine zufällig gewählte Homezone gerichtet. Danach beginnt der Countdown. Dieser besteht aus einer Kette von Nachrichten, die durch `timer:send_after` nacheinander ablaufen. Dabei wird jedesmal ein Text auf dem Client ausgegeben. Der Beginn einer Runde wird am Ende einer bisherigen Runde ausgelöst oder von der Funktion, die den Serverzustand initial bestimmt.

Bewertung

Bemerkenswert ist, dass der Client nur als Aus-/Eingabemedium für den Server dient. Der Client hat sehr wenig Information. Beispielsweise kann im Client kein Zusammenhang zwischen der eigenen Spielernummer und der Highscoreliste bzw. dem eigenen Paddle hergestellt werden. Deshalb ist es nicht möglich, intelligente automatische Spieler zu implementieren. Diese sind sozusagen blind, da sie nicht wissen, welches Element der *Paddles*-Liste sie bewegen. In unserer Anwendung ist das nicht problematisch, in entsprechenden Applikationen muss so etwas aber berücksichtigt werden.

Eine der größeren Schwächen des Spiels ist, dass es synchron läuft. Ein neuer Gametick beginnt also erst, wenn der alte abgeschlossen ist. Bei langsamen Netzsegmenten oder hoher CPU-Last auf einem Client beginnt das Spiel zu ruckeln. Es wäre aber möglich, einen Gamtick beginnen zu lassen, bevor der alte vollständig abgeschlossen ist. Spieler, die nicht rechtzeitig antworten, werden interpoliert oder gar nicht bewegt. Damit sinken die Chancen eines solchen Spielers, die anderen können jedoch ruckelfrei spielen. Kommerzielle Spiele betreiben einigen Aufwand, um Spieler mit langsamer Anbindung mitspielen zu lassen.

Ebenfalls verbesserungsfähig ist das Punktesystem des Pongspiels. Im Moment können Mitspieler nur schlechter werden, der beste Spieler hat immer null Punkte. Denkbar wäre, die Ballverluste in ein Verhältnis zu Ballkontakten des eigenen Paddles zu setzen.

3.6.5 Tron

Tron ist ein einfaches Geschicklichkeitsspiel für beliebig viele Mitspieler. Das Spielfeld besteht aus einem Raster einstellbarer Größe, auf dem jeder Spieler eine Schlange (dargestellt durch einen Pfeil) bewegen kann. Die Bewegungsmöglichkeiten sind auf die vertikalen und horizontalen Richtungen eingeschränkt. Das Ziel ist es, die Schlange so lange wie möglich über das Raster zu bewegen, ohne mit dem Schlangenkopf (Pfeilspitze) den eigenen Schwanz (Pfeil), den eines Mitspielers oder die Begrenzungen des Spielfeldes zu berühren.

Am Beginn eines Spiels informiert der Tronserver alle Tronclients über einige Konfigurationseinstellungen, die die grafische Darstellung der Spielfläche betreffen. Dann beginnt die erste Spielrunde. Sowohl der Server als auch die Clients befinden sich nun im Zustand `idle` und warten darauf, dass alle Mitspieler auf den Start-Knopf drücken. Nach dem Drücken wechselt der jeweilige Client in den Zustand `waiting`. Sobald alle Spieler gestartet haben, geht der Server ebenfalls in den Zustand `waiting` und beginnt, einen dreisekündigen Countdown zu zählen, damit sich die Spieler auf den Spielbeginn einstellen können. Sobald die Null erreicht ist, beginnt das Spiel und sowohl der Server als auch die Clients gehen in den Zustand `running`. In diesem Zustand bleiben sie, bis ein Spieler gewinnt, weil alle anderen gegen eine Wand oder die Schlange eines Mitspielers gestoßen sind. In diesem Fall wird der Name des Gewinners allen Mitspielern bekannt gegeben und der überlebende Spieler bekommt den im oberen Teil des Spielfensters angezeigten Bonus auf sein Punktekonto gutgeschrieben. Die Punkte aller Spieler werden in der Spielerliste angezeigt.

Die Synchronisation des Spiel geschieht über Gameticks. Der Server sendet in bestimmten Abständen eine Nachricht an alle Clients. Diese antworten dann mit einer Nachricht, welche die Aktion enthält, die in diesem Tick durchgeführt werden soll, z.B. ein Drehen nach links oder rechts. Sobald alle Antworten beim Server eingetroffen sind, wird der Zustand des Spiels um einen Schritt weiterberechnet. Dabei werden dann die Aktionen der Spieler ausgewertet und fließen in die Berechnung ein. Des Weiteren werden dann alle Kollisionstests durchgeführt und geprüft, ob das Spiel zu Ende ist, weil nur einer der Spieler im Spiel geblieben ist.

Nachrichten-Referenz – Tron

TC → **TS** {tron, join, *Username*, *Pid*} :
Der Tronclient *Pid* des Benutzers *Username* meldet sich beim Tronserver an. Dies geschieht zum Beginn des Spiels.

TC → **TS** {tron, leave, *Username*} :
Der Tronclient des Benutzers *Username* meldet sich ab, nachdem der Benutzer „Close“ gedrückt hat.

TC → **TS** {tron, ready, *Username*} :
Diese Nachricht wird an den Server geschickt, wenn im Zustand `idle` auf „Start“ gedrückt wurde.

TS → **TC** {tron, ready} :
Alle Spieler haben auf den „Start“-Knopf geklickt und der Zustand des Spiels wechselt auf `waiting`.

TS → **TC** {tron, tick, *Ticks*, *Bonus*} :
Ein Gametick ist verstrichen. Die Anzahl aller Gameticks und der aktuelle Bonus werden übergeben, um diese im Spielfenster anzuzeigen.

TC → **TS** {tron, tack, *Username*, *Action*} :
Dies ist die Anwort auf einen Gametick. Sie gibt an, welche Aktion der Benutzer in diesem Gametick durchgeführt hat. Dies kann eine Drehung nach links (`left`), nach rechts (`right`) oder keine Drehung sein (`none`).

TS → TC {tron, players, *Players*} :
Der Zustand aller Spieler wird übermittelt, um sie in das Spielfeld zu zeichnen. *Players* ist eine Liste von `player`-Records.

TS → TC {tron, scorelist, *Players*} :
Der Zustand aller Spieler wird übermittelt, um die Spieler- und Punkteliste neu zu zeichnen. *Players* ist eine Liste von `player`-Records.

TS → TC {tron, message, *Message*} :
Der Tronserver schickt den String *Message*, den der Client anzeigen soll.

TS → TC {tron, reset} :
Eine Runde wurde beendet und der Client soll in den Zustand `idle` zurückkehren.

Bewertung

Der Tronclient hat zwar den Zugriff auf alle Spielerinformationen, die auch der Server besitzt, denn die komplette Spielerliste wird an den Client geschickt. Dies geschieht aber nur der Einfachheit halber. Der Client braucht nur die zur Darstellung wichtigen Informationen über die Spieler, also die Koordinaten, Farben, Punktzahlen und Namen der Spieler.

Leider wird das Spiel bei vielen Spielern, einer großen Rastergröße und längerer Spielzeit sehr langsam, da die Kollisionserkennung immer aufwendiger wird, je länger die Schlangen der Mitspieler werden. Auch die Netzwerklast nimmt dann rapide zu, so dass das Spiel über eine langsame Verbindung nicht mehr spielbar ist. Die Spielerzustände ließen sich effizienter repräsentieren, indem nicht jeder Rasterpunkt, sondern nur Anfangs- und Endpunkt einer Strecke gespeichert würden. Dies würde den Speicherbedarf und die Netzlast drastisch reduzieren, allerdings die Kollisionserkennung erschweren.

3.7 Die About-Box

About-Boxen finden sich in vielen Applikationen. Sie bieten den Programmierern eine Möglichkeit, ihre Namen der Öffentlichkeit zu zeigen.

Die Gorn About-Box demonstriert, wie sich in Erlang Binärdateien sehr einfach lesen, transportieren und schreiben lassen. Dazu bietet der Aboutserver eine Reihe von kleinen Bildern an, die die Aboutclients in unterschiedlichen Abständen abfragen und anzeigen.

Nachrichten-Referenz – About-Box

C → S {about, image, *Pid*} :
Ein Client fordert ein weiteres Bild an, welches zufällig aus der Liste der verfügbaren Bilder gewählt wird. Der Server hat alle Bilder zum Start des Gorn-Servers bereits gelesen und muss nur noch ein Listenelement auswählen und an die angegebene *Pid* schicken.

S → C {about, image, *Filename*, *Binarydata*} :
Der Server liefert ein Bild aus. Das Tupel enthält den Dateinamen des Bildes und die eigentlichen Daten. Auf dem Client werden die Daten in eine temporäre Datei geschrieben und das Grafiksystem wird angewiesen, das Bild in einem Canvas anzuzeigen. Dies ist ein wenig umständlich, da es nicht möglich ist, das Bild in Form von Binärdaten direkt an das Grafiksystem zu übergeben. So muss der Client Sorge tragen, dass die temporäre Datei wieder entfernt wird.

3.8 Die Bibliothek `libgorn`

In den Applikationen des Gorn-Systems gibt es natürlich wiederkehrende Probleme, deren Lösungen in einer Bibliothek gesammelt werden können. Zu diesem Zweck wurde die `libgorn`-Bibliothek

eingeführt, die in vielen Linklisten der Applikationen vorkommt.

Eine kleine Gruppe von Funktionen dient dazu, das „Look & Feel“ einheitlich zu halten. Das sind Funktionen, die Schriftarten vorgeben oder einen String zusammensetzen, der als Fenstertitel verwendet werden kann. Auch ein zusammengesetztes grafisches Element, der Schriftscroller, ist hier implementiert.

Die wichtigste Gruppe von Funktionen in der Bibliothek beschäftigen sich mit Zufallszahlen. Es gibt Funktionen, die zufällige Fließkomma- oder Integerzahlen mit gegebenen Grenzen erzeugen. Dabei wird das `random`-Modul benutzt, das zu ERLANG gehört. Allerdings wird bei Bedarf der Seed automatisch festgelegt. So ist es in einer Applikation einfacher, den Zufallszahlengenerator zu benutzen, auch an Stellen, an denen nicht klar ist, ob der Generator schon initialisiert wurde oder nicht.

Zum Initialisieren des Generators kommt eine etwas kompliziertere Funktion zum Einsatz. Das Problem ist, dass der ERLANG Zufallsgenerator aus drei Zahlen (Seedwerten) initialisiert wird und bei gleichen Seedwerten auch die gleichen Zahlenketten erzeugt. Es ist also sehr wichtig, immer wechselnde Seedwerte zu erzeugen. Zusätzlich muss in jedem Prozess der Zufallsgenerator neu initialisiert werden, da der Seed von ERLANG prozesslokal gehalten wird. Ein klassischer Ansatz ist, die aktuelle Zeit in den Seedwerten zu benutzen. Die `localtime`-Funktion liefert aber nur sekundengenaue Zeiten. Werden viele Prozesse schnell hintereinander gestartet, haben alle identische Seedwerte und damit auch die selben Zufallszahlen. Dieses Problem trat beispielsweise bei den automatischen Pongspielern auf.

Das `timer`-Modul von ERLANG bietet eine Funktion `tc`, die eine Funktion ausführt und die verstrichene Zeit in Mikrosekunden zurückliefert. Zum Erzeugen einer der drei Seedwerte wird nun eine Funktion aufgerufen, die zwei kurze (wenige Millisekunden) `sleep`-Aufrufe absolviert. Die Idee ist, dass Zeitmessungen in diesem Bereich sehr ungenau sind, gerade wenn durch viele neu gestartete Prozesse viele Taskwechsel auftreten. Es hat sich gezeigt, dass so Werte erzeugt werden können, die sich in den Prozessen unterscheiden. Für die anderen beiden Seedwerte werden die Minuten und Sekunden der aktuellen Uhrzeit verwendet. Die Funktion `check_random_seed` stellt sicher, dass der Zufallszahlengenerator in jedem Prozess nur einmal initialisiert wird und tut dies auch, wenn es nötig ist.

4 Fazit

4.1 Die Programmiersprache Erlang

Die Erlang Laufzeitumgebung ist sehr praktisch, da Codeteile und Funktionen außerhalb ihres Einsatzgebietes getestet werden können. Dies ist sehr wichtig, da das Typsystem keine Überprüfungen zur Kompilierzeit ermöglicht. Deswegen ist es von großer Bedeutung, alle Codeteile genau zu testen. Dies ist nicht einfach, da alle Zustände der Applikation durchlaufen werden müssen. Das dynamische Typsystem zeigt sich auch an anderen Stellen als negativ. Funktionen können entweder mit einer `-import()`-Direktive in den Namensraum eines Moduls importiert oder mit dem Modulnamen voll qualifizert aufgerufen werden. Bei beiden Methoden findet aber keine Überprüfung der Existenz des Moduls oder der Funktion statt, so dass sich schnell unbemerkt Fehler einschleichen. Allerdings ist es dadurch möglich, Modulnamen dynamisch zu erzeugen.

Der Kommunikationsmechanismus von `!`-Operator und `receive`-Blöcken ist sehr schön zu benutzen und von großer Klarheit. Auch das Patternmatching über Funktionsköpfe und empfangene Nachrichten ist sehr praktisch. Es zeigt sich dabei, dass oft Paare von Codestellen existieren, in denen eine Nachricht erzeugt bzw. verbraucht wird. Wenn sich beim Programmieren hier ein kleiner Schreibfehler einschleicht, artet die Suche danach schnell in ein aufwendiges Debugging aus. Auch die Vollständigkeit und Konfliktfreiheit einer Funktionsimplementierung mit verschiedenen Pattern im Kopf kann nicht statisch geprüft werden und zeigt sich erst zur Laufzeit in teilweise recht undeutlichen `badmatch` Fehlern.

Im Gorn-System kommt noch erschwerend hinzu, dass sehr häufig Codeteile auf anderen Knoten gestartet werden, die Rückgabewerte der aufgerufenen Funktionen aber nicht von Interesse sind. Deswegen werden Fehlermeldungen, die in Rückgabewerten verschlüsselt sind, verschluckt, und abstürzende Clientapplikationen müssen durch „print-debugging“ repariert werden. Allerdings hat erst die Möglichkeit des dynamischen Nachladens von Codeteilen diesen Mechanismus möglich gemacht. Wie Gorn gezeigt hat, ist mit Erlang echtes „Application Service Providing“ möglich: Nutzer haben nur absolut notwendige Codeteile lokal vorrätig, während die eigentliche Applikationslogik zentral verwaltet werden kann und erst bei Bedarf auf die Clients übertragen wird. Die Berechnungen selber finden dann auf dem Client statt, so dass die Rechenlast beim Client verbleibt.

Obwohl die Kommunikation zwischen Programmteilen in Erlang sehr einfach ist, ergeben sich schnell wieder Standardprobleme, die nicht gelöst werden. Hier wäre eine Bibliothek oder zumindest eine Sammlung von „Design Pattern“ sehr hilfreich. Das erste Standardproblem ist die Identifikation des Absenders einer Nachricht. Diese ist natürlich nicht immer nötig, man denke an Events, die aus dem Grafiksystem kommen. In einem System mit verschiedenen Usern ist es jedoch schwierig, Sicherheit herzustellen. In Gorn wurde das Problem durch eine Art „Session Cookie“ gelöst, allerdings ist es natürlich immer mit Aufwand verbunden, eine eigene Lösung zu implementieren. Es ist in Erlang relativ einfach, Nachrichten in Abhängigkeit vom aktuellen Zustand zuzulassen oder zu verbieten, eine Abhängigkeit vom Absender ist aber nicht möglich. Sollen nur zwei Prozesse immer wechselseitig kommunizieren, ergibt sich eine Art verbindungsorientierte Kommunikation, die über der „Paketorientierten“ des Messagepassing steht. Erlang bietet keine Hilfe an, die Existenz des Kommunikationspartners zu überprüfen. Als Folge daraus, müssen Ping/Pong Mechanismen implementiert werden, die regelmäßig das Vorhandensein der Gegenstelle überprüfen. Im Gorn-System wird das beispielsweise gemacht, um das Verschwinden einer Gornbar zu erkennen, welche nicht ordnungsgemäß beendet wurde (z.B. durch Abschießen der Erlang Shell).

Etwas umständlich ist das Aufsetzen der Erlang Laufzeitumgebung. Damit Knoten überhaupt kommunizieren können, muss die Namensauflösung der Rechner auf allen Knoten in beide Richtungen vollständig und korrekt sein. Dies ist in großen und professionell administrierten Netzen unproblematisch, aber bei vielen kleinen Installationen bei Privatleuten oder Verbünden von tragbaren Computern eher mühselig zu erreichen. Zusätzlich hat sich gezeigt, dass Erlang Netzwerkverbindungen auch automatisch zwischen Clients aufbaut, die sich auf einen gemeinsamen Server verbunden haben. Dies ist für die Kommunikationsgeschwindigkeit ideal, da benachbarte

Knoten direkt kommunizieren können, ohne Umwege in Kauf zu nehmen. Der negative Effekt ist allerdings, dass es nicht möglich ist, ERLANG-Knoten durch maskierende Firewalls (NAT, Network Address Translation) uneingeschränkt zu betreiben. So ist z.B. GORN nur durch eine solche Firewall zu benutzen, wenn die Spiele nicht verwendet werden. Diese funktionieren nicht, und es werden keine Fehlermeldungen erzeugt.

Weiterhin bemerkenswert ist die Garbage Collection des ERLANG-Systems. Im normalen Betrieb ist vom Garbage Collector nichts zu bemerken. Lediglich in den rechenintensiveren Spielen macht er sich nach einer Zeit bemerkbar. Das Spiel Tron beispielsweise wird merklich langsamer, da die Linien in immer länger werdenden Listen verwaltet werden. Das Spiel Pong setzt einige Rechenleistung voraus, da die ganze berechnete Welt in dynamisch erzeugten und ständig manipulierten Listen gehalten wird.

Von großer Nützlichkeit ist der ERLANG Debugger. Dieser ermöglicht die Untersuchung eines laufenden ERLANG Prozessgeflechts. Der Debugger selber unterstützt alle gängigen Techniken, wie Breakpoints und Step-by-Step Ausführung. Leider ist es uns nicht gelungen, die Clientapplikationen des GORN-Systems im Debugger zu untersuchen. Vermutlich ist uns dort das vollständig dynamische Nachladen und Distributieren des Binärcodes in die Quere gekommen.

Ein etwas größerer Schwachpunkt von ERLANG ist die Graphikschnittstelle Tk. Diese ist durchaus verbreitet, aber eigentlich nicht besonders schön. Tk lässt sich relativ einfach programmieren, das Zusammenbauen einer grafischen Nutzerschnittstelle resultiert jedoch in sehr viel schlecht zu testendem Code, der zudem kaum zu warten ist. Wurde in einem Fenster ein System aus Frames aufgebaut, kann es sehr aufwendig sein, einen Knopf an der richtigen Stelle hinzuzufügen. Funktioniert eine Anhäufung grafischer Elemente nicht wie erwartet, ist z.B. ein Label nicht sichtbar, obwohl es da sein müsste, ist es schwierig, das Problem zu isolieren. Beim Erzeugen der grafischen Elemente gibt es eine große Menge von Optionen, die alle erst zur Laufzeit ausgewertet werden, so dass sich dementsprechend Fehler nicht statisch finden lassen. Es passiert schnell, dass die Option zum Einschalten einzelner Eingabeevents falsch geschrieben wird und deswegen viele Events nicht auftreten. Weiterhin muss dem Programmierer bewusst sein, dass sich mit dem `GS` einfach Oberflächen erstellen lassen, diese aber noch keineswegs benutzerfreundlich sind. So ist zum Beispiel ein Springen zwischen den einzelnen Elementen per Tastendruck, ein erwartetes Standardverhalten, jedesmal per Hand zu implementieren.

Eine GORN-Applikation lässt sich nur zweimal aus der Gornbar starten. Sobald ein drittes Fenster geöffnet wird, wird das Erste automatisch geschlossen. Den Grund für dieses Problem konnten wir nicht abschließend klären, es liegt aber vermutlich an dem Mechanismus, mit dem wir dynamisch Module nachladen. ERLANG verwaltet zwei Versionen der geladenen Module (*old* und *new*), damit Funktionen, die auf ein geladenes Modul zugreifen, trotz des nachgeladenen Moduls weiterarbeiten können.

Für GORN, insbesondere für die Spiele, wäre es nett gewesen, wenn einfache Pieptöne erzeugt werden könnten. Dafür ist in der Standardbibliothek nichts vorgesehen.

Die Syntax von ERLANG ist minimalistisch. Obwohl sie logisch ist, ergeben sich doch einige Kritikpunkte. Zunächst sind die bedingten Ausdrücke syntaktisch zu kompliziert. If/else Konstrukte sehen ungewohnt aus, da im else-Teil häufig eine Bedingung bzw. ein Pattern wie `true` oder `_` („don't care“) vorkommt. Der Verkettungsoperator „`,`“, der sowohl für Listenkonstruktionen als auch für die Verkettung von Funktionsanwendungen benutzt wird, unterstützt kein leeres Element, so dass das letzte Element nie von einem Komma gefolgt werden darf. Dies ist zwar durchaus logisch, stellt sich aber als primäre Fehlerquelle heraus, wenn Teile einer Liste kurzzeitig auskommentiert werden (beim neuen letzten Element muss das Komma entfernt werden) oder eine Liste nachträglich erweitert wird (beim nun nicht mehr letzten Element muss ein Komma hinzugefügt werden). Ähnlich verhält es sich mit dem Verkettungsoperator für Alternativen, dem Semikolon. Auch dieses muss immer beim letzten Element weggelassen werden.

Records sind eine von uns sehr viel genutzte Möglichkeit, dem dynamischen Typsystem schon

zur Kompilierzeit etwas mehr Strenge zu geben. Beim Benutzen von Records kann wenigstens der Zugriff auf ein nicht existierendes Element statisch erkannt werden. Viele Tippfehler lassen sich so früh ausmerzen. Allerdings ist die Syntax zum Zugriff auf einzelne Elemente des Records sehr umständlich, da jedesmal der Recordtyp mit angegeben werden muss. Werden Records verschachtelt, wird die Syntax noch komplizierter, da der Zugriff auf das äußere Element geklammert werden muss. All das ist durchaus logisch, aber nicht praktisch.

Weiterhin unterstützt ERLANG kleine lokalen rekursiven Funktionen. Eine lokale Funktion, die mittels eines Lambda Ausdrucks (`fun(...)` → `... end`) definiert wurde, kann sich selbst nicht aufrufen, da der Funktionsname im Funktionsrumpf nicht bekannt ist. Daher müssen rekursive Hilfsfunktionen immer modulglobal definiert werden.

Noch zu erwähnen ist die Dokumentation des ERLANG-Systems und der Standardbibliotheken. Wir haben die HTML Version in der Entwicklung benutzt. Leider ist diese nicht durchsuchbar, so dass eine Stichwortsuche nicht möglich ist. Die Funktionen sind innerhalb einer Moduldokumentation nicht alphabetisch sortiert. Weiterhin ist die Dokumentation nicht immer vollständig und an manchen Stellen sogar falsch. Ein Studium der Quellen ist manchmal unumgänglich. Da z.B. im `GS` nicht alle verfügbaren Konfigurationsoptionen genannt sind, müssen einige direkt in den Quellen gesucht werden.

Weniger mit ERLANG als eher mit Gewohnheiten hat das funktionale Programmieren an sich zu tun. Es ist im ersten Moment etwas gewöhnunsbedürftig, Schleifen, wie z.B. die Eventbehandlung, als tailrekursive Funktion zu formulieren. Später muss darauf geachtet werden, die Tailrekursivität nicht wieder zu zerstören.

4.2 Bewertung der Arbeitsergebnisse

Obwohl GORN gut benutzbar ist, wären noch einige Funktionen zu verbessern und Unstimmigkeiten zu beseitigen. Einige der Probleme betreffen die grafische Oberfläche. Listeneinträge, z.B. in der Benutzerliste der Gornbar, bleiben nur so lange markiert, bis die Liste neu aufgebaut wird. Leider wird dies von Tk nicht automatisch behandelt, sondern müsste in Handarbeit nachgerüstet werden. Ebenfalls per Hand muss das Wechseln des Eingabefokus programmiert werden. Das Löschen von Benutzerdaten sollte mit einer Rückfrage abgesichert werden.

Der oben beschriebene Ping/Pong-Mechanismus ist nicht in allen Applikationen konsequent implementiert, so dass die Spiele hängen bleiben können, wenn eine Clientapplikation abstürzt, ohne sich zuvor beim Spiel abzumelden.

Wir haben uns bemüht, ein gewisses Maß an Sicherheit in das GORN-System zu integrieren. Dennoch ist das System nicht sicher. Das liegt zum einen am Prinzip des Nachrichtenaustauschs in ERLANG, aber auch an Implementierungsmängeln und daran, dass interne Nachrichten nicht authentifiziert werden.

Es hat sich gezeigt, dass wir wiederkehrende Problemstellungen verschieden gelöst haben. So haben sich beispielsweise für Benutzerlisten in Applikationen zwei verschiedene Herangehensweisen herauskristallisiert. In einigen Applikationen wird die Liste zentral gehalten und bei einer Änderung komplett auf alle Clients übertragen. Im Chat jedoch wird die Liste zwar zentral auf dem Chatserver verwaltet und einem neuen Client komplett mitgeteilt. Spätere Änderungen werden aber einzeln bekannt gegeben, und jeder Client pflegt seine eigene Nutzerliste.

Es hat sich als hilfreich erwiesen, das kommunikationsbasierte System durch die versendeten Nachrichten zu spezifizieren und zu dokumentieren.

ERLANG ist tatsächlich gut geeignet, Programme im Rapid-Prototyping Verfahren zu entwickeln. Dies ist ein erklärtes Ziel der Entwickler von ERLANG. Nachdem wir erfolgreich einen Prototypen erstellt haben, ließe GORN sich leicht und relativ schnell in einer anderen Programmiersprache implementieren.

ERLANG eignet sich hervorragend, um nebenläufige und vor allem verteilte Systeme zu realisieren. Die flexible Laufzeitumgebung ermöglicht die Implementierung sehr modularer Systeme. GORN hat gezeigt, dass sich sehr dynamische Applikationen, verteilt auf mehrere Rechner, damit umsetzen lassen.

A Benutzerhandbuch

Dieser Abschnitt beschreibt die Installation des GORN-Systems sowie die Benutzung der einzelnen Elemente. Dabei wird zu jedem Teil der Applikation beschrieben, was Sie sehen können und welche Aktionen möglich sind.

A.1 Der GORN Server

Die Installation des GORN-Servers umfasst zwei Schritte:

- Kompilieren der ERLANG-Module,
- Initialisierung der Datenbank.

Zum Kompilieren der Module existieren in allen Unterverzeichnissen von `src/` `Makefile`-Dateien. So kann das ganze Projekt mit

```
make
```

im `src/` Verzeichnis übersetzt werden. Soll nur der Serverteil übersetzt werden, ist `make` im `src/gorn/` Verzeichnis auszuführen.

Zwei Skripte zum Initialisieren der Datenbank stehen zur Verfügung. Zum einen kann das Skript

```
./make-db
```

benutzt werden, wenn auf dem Rechner die ERLANG Option `-name` funktioniert. Andernfalls sollte das Skript

```
./make-db-sname
```

benutzt werden. Dieses Skript wird die Datenbank auf der aktuellen Node im Verzeichnis `db/` anlegen. Stellen Sie sicher, dass dieses Verzeichnis nicht existiert bzw. leer ist, bevor Sie eines der Skripte ausführen. Um eine existierende Datenbank erst zu löschen, bevor sie neu erzeugt wird, stellt das `Makefile` in `src/gorn/` zwei Targets zur Verfügung:

```
make db-reinit
```

bzw.

```
make db-reinit-sname
```

Nachdem die Datenbank initialisiert wurde, kann der GORN-Server mit dem Kommando

```
./gorn
```

gestartet werden, wenn ERLANG auf diesem Rechner die Option `-name` unterstützt. Andernfalls ist wiederum

```
./gorn-sname
```

zu benutzen. Dies wird den Server starten und dieser ist nun bereit, Verbindungen von Gornbars anzunehmen. Der Prozess hat sich mit dem Namen `gorn` registriert.

A.2 Die Gornbar

Zunächst muss die Clientapplikation kompiliert werden. Dazu muss lediglich der Befehl

```
erlc gornbar.erl
```

aufgerufen werden. Alternativ steht auch hier ein `Makefile` zur Verfügung.

Zum Starten der Gornbar sollte eines der Skripte

```
./gornbar
```

oder

```
./gornbar-sname
```

verwendet werden, je nachdem ob die lokale ERLANG Installation mit der Option `-name` oder `-sname` funktioniert. Die Skripte benötigen ein Kommandozeilenargument, welches aus dem Namen des Rechners besteht, auf dem der GORN-Server läuft. Im Falle des Skriptes `gornbar` muss dieser Name voll qualifizert sein. Dieses Argument wird in einer Datei

```
gorn-server
```

zwischengespeichert, so dass es bei weiteren Aufrufen nicht noch einmal mit angegeben werden muss. Ein vollständiger Aufruf kann also so aussehen:

```
./gornbar aeneas.cs.tu-berlin.de
```

Wenn die Gornbar gestartet wird, erscheint am Anfang eine Ausgabe wie

```
gornbar: Configured for server 'gorn@aeneas.cs.tu-berlin.de'.
gornbar: ping: pong.
```

Dabei bedeutet das `pong`, dass die Kommunikation des ERLANG-Systems funktioniert. Erscheint hier `pang`, ist eine Kommunikation nicht möglich und GORN kann nicht funktionieren.

Nach dem Starten der Gornbar sind drei Knöpfe zu sehen. Der „Away“-Knopf hat hier noch keine Funktion und mit dem „Close“-Knopf kann GORN jederzeit verlassen werden. Um sich am GORN-System anzumelden, betätigen Sie den „Login“-Knopf und geben Ihren Usernamen und Ihr Passwort ein.

Nach dem Einloggen sehen Sie die dreigeteilte Gornbar. Links sind untereinander alle Applikationen aufgelistet, die im GORN-System verfügbar sind. Rechts befindet sich eine Liste aller User in dieser Arbeitsgruppe. Alle User, die ein Sternchen vor dem Namen haben, sind gerade eingeloggt und arbeiten mit GORN. Alle User, die einen Unterstrich vor dem Namen haben, sind gerade eingeloggt, haben sich aber als „away“ gemeldet, sind also gerade nicht direkt mit GORN beschäftigt. Sie selber können sich durch Druck auf den „Away“-Knopf als abwesend kennzeichnen, der dann verfügbare „Back“-Knopf meldet Sie wieder zurück. Alle anderen User sind gerade nicht eingeloggt.

Im unteren Abschnitt der Gornbar befindet sich ein Textausgabefenster, in dem wichtige Nachrichten erscheinen. Dort erscheint beispielsweise eine Ausgabe, wenn Sie eine neue Email erhalten.

Mit dem „Logout“-Knopf können Sie sich aus dieser Arbeitsgruppe abmelden, die Gornbar wird aber nicht beendet, so dass Sie sich wieder einloggen können.

A.3 Email

Hinter dem „Mail“-Knopf der Gornbar verbirgt sich ein kleines Emailsystem, welches elektronische Nachrichten GORN-intern mit Usern der Arbeitsgruppe austauschen kann. Empfänger müssen nicht eingeloggt sein.

Das Mail Center besteht aus einer Liste von Nachrichten, die zum Lesen ausgewählt werden können. Eine ausgewählte Nachricht kann beantwortet („Reply“-Knopf), weitergeleitet („Forward“-Knopf) oder als gelöscht markiert werden („Delete“-Knopf). Der „Expunge“-Knopf entfernt die als gelöscht markierten Nachrichten tatsächlich aus dem Mailsystem.

Als gelöscht markierte Nachrichten sind mit einem „D“ markiert, neue Nachrichten mit einem „N“. Sobald eine Nachricht markiert wird, z.B. um sie zu lesen, verschwindet das „N“ und ein Unterstrich wird angezeigt.

Zum Erstellen einer neuen Nachricht dient der „Compose“-Knopf. Das erscheinende Fenster ist identisch mit dem für beantwortete oder weitergeleitete Nachrichten. In die erste Zeile, die mit „To:“ gekennzeichnet ist, ist die Liste der Empfänger einzutragen. Dies kann ein einzelner Username sein (Vorsicht: Kleinschreibung) oder eine kommagetrennte Liste von Usernamen.

In der nächsten Zeile „Subject:“ wird ein kurzer Stichpunkt genannt um den es in der Nachricht gehen soll. Dieser ist auch in der tabellarischen Übersicht der Nachrichten zu sehen. In das große Feld unten wird die eigentliche Nachricht eingetragen. Dies sind beliebig viele Zeilen Text. Zum Senden der Nachricht ist dann einfach der „Send“-Knopf zu betätigen. Kann die Nachricht nicht zugestellt werden, weil ein Empfänger nicht existiert, erzeugt das Mailsystem eine Antwort, die Ihnen das sofort mitteilt.

Alle Aktionen können durch das Betätigen des „Close“-Knopfes abgebrochen werden.

A.4 Chat

Mit dem Betätigen des „Chat“-Knopfes begeben Sie sich in den Chatkanal des Gorn-Systems. Auf der rechten Seite befindet sich eine Liste der teilnehmenden Nutzer. Im großen Feld links findet das Chatgeschehen statt. Ihre Eingaben tätigen Sie in der Zeile unten und schließen diese dann mit der `Enter`-Taste ab.

Wenn ein Nutzer etwas schreibt, wird der Name dem Text farbig vorangestellt und im Chatfenster ausgegeben. Auch ein „Agieren“ im Chat ist möglich. Dazu wird der Eingabe ein `/me` vorangestellt und eine Aktion in der dritten Person beschrieben. Dies wird im Chat wiederum farbig gekennzeichnet. Verlässt ein User den Chat oder ein neuer User betritt diesen, erscheint eine andersfarbige Anzeige und die Liste auf der rechten Seite verändert sich.

Der „Info“-Knopf fordert vom Server genaure Informationen zu den in der Userliste selektierten Usern an. Das beschränkt sich im Moment auf den vollen Realnamen. Mittels des „Close“-Knopfes können Sie den Chat verlassen.

A.5 Talk

Talk dient zur direkten Kommunikation von zwei Teilnehmern. Wenn Sie mit jemandem aus der Arbeitsgruppe talken möchten, wählen Sie den Namen aus der Userliste in der Gornbar aus und betätigen den „Talk“-Knopf. Der User muss gerade eingeloggt sein. Wenn der User den Talk annimmt, tippen Sie Ihren Text im oberen Fensterteil ein. Im unteren erscheint der Text des Gesprächspartners.

Sie können auch die Talkapplikation starten, ohne einen User ausgewählt zu haben. Geben Sie dann in dem Eingabefeld links oben den Usernamen des gewünschten Gesprächspartners ein und betätigen den „Connect“-Knopf.

Wenn Sie selber angetalkt werden, erscheint ein kleines Fenster mit der Frage, ob Sie ein Gespräch mit einer Person annehmen wollen. Sie können das durch einen der entsprechenden Knöpfe annehmen oder ablehnen. Wenn Sie den Talk annehmen, erscheint das gewohnte Talkfenster, welches schließlich jederzeit und wie gewohnt durch den „Close“-Knopf beendet werden kann. Beim Gesprächspartner verschwindet das Fenster dann ebenfalls sofort.

A.6 Admin

Die Adminapplikation dient zum Verwalten der Nutzer in dieser Arbeitsgruppe. Ein normaler Nutzer kann diese Applikation nur benutzen, um seinen Realnamen oder sein Passwort zu ändern. Dazu ist in der erscheinenden Liste der User der eigene Username zu wählen und mittels des „Edit“-Knopfes die entsprechende Bearbeitungsmaske zu öffnen. Nachdem dort die Änderungen vorgenommen wurden (Achtung: Das Passwort muss zweimal eingegeben werden, um Irrtümer

auszuschließen), können die Änderungen durch den „Save“-Knopf abgespeichert und aktiviert werden. Wiederum dienen die „Close“-Knöpfe zum vorzeitigen Beenden der verfügbaren Aktionen.

Spezielle privilegierte Benutzer können User anlegen sowie alle User bearbeiten. Zum Anlegen eines Users dient die Maske, die sich hinter dem „New“-Knopf der Adminapplikation verbirgt. Nachdem alle Angaben gemacht wurden, können diese durch „Save“ aktiviert werden und der neue User kann sich anmelden.

Das Löschen von Benutzern ist ebenfalls den privilegierten Benutzern vorbehalten. Dazu ist in der Userliste der Adminapplikation der entsprechende Nutzer auszuwählen und der „Delete“-Knopf zu betätigen.

In einem frisch installierten GORN-System existieren einige normale Benutzer (raimi, ilu, mgrabmue, petra) ohne Passwort und ein privilegierter Benutzer `root` mit dem Passwort `root`. Dieser soll als Ausgangspunkt für die eigene Benutzerverwaltung dienen. Zunächst sollte das Passwort geändert und die Beispielbenutzer gelöscht werden. Danach können neue Nutzer eingetragen werden.

A.7 Spiele: GORNtris, Tron, Pong

Die Spiele verbergen sich hinter den gleichnamigen Knöpfen in der Gornbar und benutzen das gleiche Verfahren zum Eröffnen von Spielen oder der Teilnahme.

Zunächst erscheint ein Fenster, in dem ein existierendes Spiel in einer Liste ausgewählt werden kann und an dem man mittels des Drucks auf „Join“ teilnimmt, wenn die maximale Anzahl von Mitspielern noch nicht erreicht ist. Während auf andere Spieler gewartet wird, wird ein kleines Fenster angezeigt, in dem die aktuelle Konfiguration des Spiels zu sehen ist. Durch Drücken des „Quit“-Knopfes können Sie jetzt noch das Spiel verlassen.

Im Spielauswahlfenster befindet sich neben der Liste der noch nicht begonnenen Spiele ein „Refresh“-Knopf, der diese Liste aktualisiert. Dies kann nötig sein, da sich die Liste automatisch nur nach fünf Sekunden Verzögerung aktualisiert.

Sie können hier ein neues Spiel eröffnen, indem Sie den „Create“-Knopf betätigen. Es folgt ein Konfigurationsdialog, der für jedes der Spiele spezifisch ist und weiter unten beschrieben wird. Ist die Konfiguration abgeschlossen, erscheint ein Fenster, in dem Sie Ihr Spiel vor dem Start verwalten können.

Zunächst kann die maximale Anzahl der möglichen Spieler verändert werden. Dazu befindet sich links in dem Fenster ein vertikaler Schieberegler. Ist diese Anzahl von Spielern erreicht, können keine weiteren Spieler mehr teilnehmen. Rechts befindet sich eine große Liste, in der die Usernamen der Spieler vermerkt sind, die derzeitig an Ihrem Spiel teilhaben wollen. Einzelne Spieler können Sie mittels des „Kick“-Knopfes ausschließen. Die Konfiguration des Spieles kann hier noch geändert werden, indem der Konfigurationsdialog durch den „Configure“-Knopf aufgerufen wird.

Sobald Sie mit der Anzahl und Art der Mitspieler zufrieden sind, können Sie durch den „START“-Knopf den Beginn das Spieles auslösen. Jetzt können keine weiteren Spieler mehr hinzukommen.

A.7.1 GORNtris spielen

GORNtris ist an das Gameboy™-Tetris™ angelehnt. Von oben fallen langsam (und schneller werdend) Steine in das Spielfeld, welche Sie mit den Pfeiltasten nach links und rechts bewegen können. Mit den Tasten 'a' und 's' kann der aktuelle Stein nach links bzw. rechts gedreht werden. Die Pfeiltaste nach unten lässt den Stein schneller nach unten sinken. Dort sollten die Steine so abgesetzt werden, dass keine Lücke ensteht. Immer wenn eine Zeile ganz gefüllt ist, wird diese aus dem Spielfeld entfernt. Schaffen Sie es 2, 3 oder sogar 4 Zeilen (eine Tetris™) gleichzeitig abzuräumen, bekommen alle anderen Spieler von unten 1, 2 oder 4 Zeilen dazugeschoben und deren Spielfeld füllt sich dementsprechend. In dem dazugeschobenen Block ist eine Spalte frei, so dass er schnell wieder zu füllen ist.

Sie verlieren ein Spiel, sobald ein neu hereinfallender Stein nicht mehr bewegt werden kann, Ihr Spielfeld also gefüllt ist. Gewinner ist der Spieler, der zuletzt übrig bleibt.

Auf der rechten Seite des GORNtris-Fensters wird eine Liste der Mitspieler angezeigt. Direkt neben der Spielfläche befindet sich in der selben Reihenfolge eine Reihe von senkrechten Balken. Diese Balken geben die Füllstände der Spielflächen der Mitspieler an. Ist einer der Balken rot und in voller Höhe, ist dieser Spieler bereits ausgeschieden.

A.7.2 GORNtris konfigurieren

Der erste Schieberegler „Initial delay for drop speed“ gibt an, wieviel Zeit (in Millisekunden) vergeht, bevor ein Stein automatisch eine Zeile tiefer rutscht. Dies ist ein Initialwert, da die Steine später immer schneller fallen. Eine kleinere Zahl hier bewirkt ein schnelleres Fallen der Steine.

Der zweite Schieberegler „Number of lines to remove between speedup“ gibt an, wieviele Zeilen insgesamt von allen Spielern abgeräumt werden müssen, bevor die Steine anfangen, wieder etwas schneller zu fallen.

Der dritte Schieberegler „Amount of speedup“ gibt an, um wieviel die Steine schneller fallen sollen, wenn der Spieler die obige Anzahl von Zeilen abgeräumt hat. Eine größere Zahl resultiert in einer höheren Beschleunigung des Spiels.

Die nächsten beiden Regler geben die Größe des Spielfeldes vor. Die Voreinstellung von 10x18 entspricht dem original Gameboy™-Tetris™.

Schließlich kann mit der letzten Einstellung gewählt werden, ob Zeilen jederzeit von anderen Spielern untergeschoben werden können („Insert at any time“), oder ob das nur möglich ist, wenn gerade ein eigener Stein abgesetzt wurde („Insert only after drop“). Dieses ist für den Spieler etwas einfacher als Ersteres.

A.7.3 Pong spielen

Pong ist ein rundenbasiertes Spiel. Sobald das Spiel gestartet wurde, beginnt ein fünfsekündiger Countdown und die Runde startet. Sie kontrollieren ein grünes Paddle an der unteren Seite des abgebildeten Polygons. Sie können es mit den Pfeiltasten oder den Tasten der Maus nach links und rechts bewegen. Der Bereich, auf dem Sie das Paddle bewegen können, ist Ihre Homezone. Aufgabe der Spieler ist es, den Ball nicht aus dem Polygon entkommen zu lassen.

Sobald ein Spieler den Ball fallen lässt, bekommt dieser Spieler einen Punktabzug und eine neue Runde beginnt. Sie können das Spiel jederzeit über den „Close“-Knopf verlassen. Ihr Paddle wird dann so groß wie Ihre Homezone, so dass alle anderen Spieler ungestört weiterspielen können.

Sollten Sie das Spiel allein spielen, steuern Sie drei Paddles in einem Sechseck.

A.7.4 Pong konfigurieren

Die erste Einstellung ist die Geschwindigkeit des Balles („Ball Speed“). Zum besseren Verständnis wird diese in Metern pro Sekunde angegeben. Das sichtbare Spielfeld hat einen Durchmesser von 1000 Metern.

Der zweite Schieberegler gibt an, wie schnell sich das Paddle bewegt. In der Voreinstellung von 100%, dauert es eine Sekunde, um das Paddle von einer Seite der Homezone auf die andere zu bewegen.

Der nächste Schieberegler „Paddle Size“ gibt an, wie groß das Paddle im Verhältnis zur Homezone ist. Je größer es ist, desto einfacher ist es, den Ball zu treffen.

Die Einstellungen „Smoothness“ bestimmt, wieviele Berechnungsschritte pro Sekunde durchzuführen sind. Ein höherer Wert bedeutet eine flüssigere Bewegung aller Elemente, aber auch eine höhere Belastung des Netzwerkes und des Rechners. Mit niedrigen Werten ist es möglich, über eine langsame Verbindung wie Modem oder ISDN zu spielen.

Letztendlich können Sie noch wählen, ob automatische Spieler, sogenannte Bots, am Spiel teilnehmen sollen. Diese verhalten sich sehr unkontrolliert, es lassen sich aber Spiele mit sehr vielen Teilnehmern simulieren. Für große und schnelle Paddles erreichen die Bots bessere Ergebnisse.

A.7.5 Tron spielen

Tron ist ebenfalls ein rundenbasiertes Spiel. Sobald alle Spieler durch Drücken der Space-Taste oder des „Start“-Knopfes ihre Bereitschaft zum Rundenbeginn bestätigt haben, startet ein kurzer Countdown und die Runde beginnt. In der Runde bewegen Sie den Kopf ihrer Schlange über ein Raster. Auf den Schnittpunkten des Rasters wendet sich die Schlange nach rechts oder links, wenn Sie vorher die rechte oder linke Pfeiltaste gedrückt haben. Die Bewegungsrichtung ist dabei immer aus der Perspektive des Schlangenkopfes zu sehen.

Ein Spieler verliert, wenn er an eine Wand oder den Schwanz einer anderen Schlange stößt. Gewinner der Runde ist der letzte Spieler, der sich noch bewegen kann. Danach beginnt eine neue Runde. Auf der rechten Seite des Tronfensters befindet sich eine Highscoreliste, auf der sich die Punktestände der Mitspieler befinden. Sie können das Spiel jederzeit durch Betätigen des „Close“-Knopfes verlassen.

A.7.6 Tron konfigurieren

Mit dem ersten Schalter kann das Spiel in den „Snake Mode“ versetzt werden, was bedeutet, dass die Schlangen eine maximale Länge haben. Normalerweise werden die Schlangen von Beginn an immer länger, so das sich das Spielfeld schnell mit Schlangen füllt.

Mit dem zweiten Schalter „Show Grid“ kann die Sichtbarkeit des Rasters, auf dem sich die Schlangen bewegen, ein- oder ausgeschaltet werden. Wenn ein großes Raster benutzt wird, sollte die Sichtbarkeit ausgeschaltet werden, um den Bildaufbau nicht zu verlangsamen. Der Schieberegler „Grid Size“ reguliert die Anzahl der Rasterlinien sowohl in waagerechter als auch senkrechter Richtung.

Der Regler „Game Delay“ gibt an, wie schnell das Spiel fortschreitet. Der Wert sollte sich an der Geschwindigkeit des Netzwerkes und der Größe des Rasters orientieren. Ein großes Raster sollte schneller gespielt werden.

A.8 About-Box

In der About-Box haben sich die Programmierer des Gorn-Systems verewigt und als Fans der britischen Komiker „Monty Python“ offenbart. Durch Klicken in eine der Bilderreihen oder durch Betätigen der Space-Taste lässt sich die About-Box wieder schließen.

Literatur

[1] ERLANG Homepage: `http://www.erlang.org`

[2] Die FAQs, insbesondere wegen des Problems mit der Namensauflösung: `http://www.erlang.org/faq/t1.html`

[3] ERLANG Dokumentation im Allgemeinen: `http://www.erlang.org/doc.html`

[4] ERLANG Bibliotheksreferenz: `http://www.erlang.org/doc/r7b/doc/applications.html` Diese Dokumentation gibt es auch zum kompletten Download zu jeder ERLANG Distribution.